Optimization of a Software Defined Radio multi-standard system using Graph Theory.

Patricia Kaiser

Université Libanaise
École Doctorale
Sciences et Technologies

Supélec

N° ordre : 2012-25-TH

THESE EN CO-TUTELLE

Pour obtenir le grade de Docteur délivré par

L'Ecole Doctorale des Sciences et Technologie
Université Libanaise
Spécialité : Mathématiques

***

SUPELEC
Spécialité : Traitement du Signal et Télécommunications
Equipe d'accueil : Institut d'Electronique et de télécommunications de Rennes
Ecole Doctorale : MATISSE
Composante universitaire : S.P.M.

Présentée et soutenue publiquement par

Patricia KAISER

Le 20 Décembre 2012

Optimization of a Software Defined Radio multi-standard system using Graph Theory

Directeurs de thèse **: Profs. Yves LOUËT et Amine EL SAHILI**

Membres du Jury

Stéphane Pérennes, Professeur ,INRIA ,Université de Nice	Rapporteur
Francis Lau Professeur,Université de Hong Kong	Rapporteur
Samer Lahoud, Maîtres de Conférences, Université de Rennes 1	Examinateur
Oussama Bazzi, Professeur, Université Libanaise	Examinateur
Ali Al Ghouwayel, Maîtres de Conférences, Lebanese International University	Examinateur
Yves Louët, Professeur ,SUPELEC	Co-Directeur de thèse
Amine El Sahili, Professeur, Université Libanaise	Co-Directeur de thèse

Dedications

To my parents

Acknowledgments

First of all, I want to thank God for giving me the patience, will, and courage to finish this thesis and has bestowed upon me his blessings all along these years.

Many have provided me with help and encouragement during this thesis and it is my pleasure to acknowledge their guidance and support.
I would like to begin by conveying my deep and sincere gratitude to my devoted supervisors, Professor Amine EL SAHILI and Professor Yves LOUET for their continuous help and understanding, and for always believing in my capabalities especially when I was lost for a while. With all their support, I started an exciting research journey and was able to draw a map of this journey, hoping that it can lead the way for useful results in the future.

I wish to express my warmest thanks to my parents who have given me endless love and support in all my decisions, and for all their understanding and patience throughout my educational life.

I am very thankful to Professor Francis LAU and Professor Stphane PERENNES for first having accepted to review my thesis, and then their precious time for reading and evaluating the work. I am grateful to Professor Oussama BAZZI for honoring me by accepting to be the chair person of my committee. I would like to further thank Dr. Samer LAHOUD and Dr. Ali AL GHOUWAYEL for accepting to participate to my committee and their valuable remarks.

I want to thank Dr. Ali AL GHOUWAYEL for having presented my supervisors one to another, thus participating to create this successful joint Ph.D work between the two universities; the Lebanese University (Hadath Campus) and Suplec (Rennes campus).

I wish to thank all my friends for their friendship and the pleasant atmosphere that they have created inside and outside the SCEE lab. I particularly want to thank my dear friend and colleague Salma BOURBIA for her major contribution to practice and improve my French language, as well as her availability and help concerning any queries or problems that I have faced in the programming field during my thesis work.

Finally, I would like to acknowledge the financial support provided by EGIDE (Centre franÃ§ais pour l'Accueil et les changes internationaux) and the AUF (Agence Universitaire de la Francophonie) during some periods of my Ph.D studies.

Patricia (Kaiser) Maatouk December 20, 2012.

Abbreviations

ADC	Analog-to-Digital Converter
AFE	Analog Front End
AGC	Automatic Gain Control
ASIC	Application Specific Integrated Circuit
B-arc	Backward hyperarc
BC	Building Cost
BFS	Breadth-First Search
BP	Break-down Part
BPF	Band Pass Filter
CC	Computational Cost
CF	Cost Function
CO	Common Operator
DAC	Digital-to-Analog Converter
DFE	Digital Front End
DFS	Depth-First Search
DFT	Discrete Fourrier Transform
DMFFT	Dual Mode FFT
DP	Duplicated Part
DSP	Digital Signal Processor
DTM	Deterministic Turing Machine
DUP	DP+UP
DUPB	DUP+BP
ER-LFSR	Extended Reconfigurable LFSR
ES	Exhaustive Search
F-arc	Forward hyperarc
FFT	Fast Fourrier Transform
FPGA	Field Programmable Gate Array
GNG	Generated Graph
GSM	Global System for Mobile communication
HC	Hamiltonian Cycle problem
IF	Intermediate Frequency
LFSR	Linear Feedback Shift Register
LNA	Low Noise Amplifier
LO	Local Oscillator
LUT	LookUp Table
MCD	Minimum Cost Design algorithm
MST	Minimum Spanning Tree

NDTM	Non-Deterministic Turing Machine
NoCs	Number of Calls
NP	NonDeterministic Polynomial time problem
OFDM	Orthogonal Frequency Division Multiplexing
P	Polynomial-time problem
PE	Processing Element
R-LFSR	Reconfigurable LFSR
RF	Radio Frequency
SA	Simulated Annealing
SAT	Satisfiability Problem
SBT	Shortest B-Tree
SDR	SoftWare-Defined Radio
SWR	SoftWare Radio
TSP	Traveling Salesman Problem
UMTS	Universal Mobile Telecommunication System
UP	Unduplicated Part
Wifi	Wireless - Fidelity
2G	2^{nd} Generation
3G	3^{rd} Generation
4G	4^{th} Generation

Math Notations

$V(G)$	Set of vertices of G
$E(G)$	Set of edges (hyperedges, arcs, or hyperarcs) of G
$v(G)$	Number of vertices in G
$e(G)$	Number of edges (or arcs) of G
$n(H)$	Number of vertices in H
$m(H)$	Number of hyperedges (or hyperarcs) in H
$G\backslash e$	edge-deleted subgraph
$G - v$	vertex-deleted subgraph
K_n	complete graph of order n
$N_G(v)$	Neighborhood of v in G
$d_G(v)$	degree of v in G
$l(P)$	length of path P
$\Delta(G)$	maximum degree of a vertex in G
$d_G(u, v)$	distance between u & v in G
$\alpha(G)$	stability number of G
$K_{r,s}$	complete bipartite graph partitioned into X,Y, with $\|X\| = r$ and $\|Y\| = s$
$C(G)$	closure of graph G
$\chi(G)$	chromatic number of G
$G(D)$	Underlying graph of digraph D
$\vec{G}$	an orientation of graph G
$N_D^-(u)$	in-neighborhood of u in D
$N_D^+(u)$	out-neighborhood of u in D
$d_D^-(u)$	in-degree of u in D
$d_D^+(u)$	out-degree of u in D
$\Delta^-(D)$	maximum in-degree of a vertex in D
$\Delta^+(D)$	maximum out-degree of a vertex in D
$S(x)$	star of center x in hypergraphs
$r(H)$	maximum size of a hyperedge in a hypergraph H
K_n^r	complete r-uniform hypergraph of order n
$T(E)$	tail set of hyperarc E
$H(E)$	head set of hyperarc E
$FS_H(v)$	Forward Star of v in H
$BS_H(v)$	Backward Star of v in H
$s^-(H)$	maximum in-size of directed hypergraph H
$s^+(H)$	maximum out-size of directed hypergraph H
$L(v)$	level of block (or vertex) v

$BF_P(e_{i_j})$	BF-reduction of e_{i_j} via P
$w(P)$	weight of path P
$w_E(x, y)$	Number of calls (or weight) on the BF-reduction (x, y) of hyperarc E
GF	Galois Field
$\mathbb{C}$	Complex Field

Contents

Part I

Résumé en français

Résumé en français

Introduction

Dans le domaine des télécommunications, ces dernières années ont connu une prolifération de normes et standards, en particulier dans les communications sans fil (téléphonie mobile, diffusion de données, réseaux locaux, etc.) pour n'en citer que quelques-uns. La nécessité d'un terminal adaptable, flexible et universel capable de prendre en compte les changements (de zones géographiques, de service, de fonctionnalités) de façon transparente pour l'utilisateur est apparue progressivement comme une nécessité.

Au début des années 90, J. Mitola a introduit la notion de radio logicielle (SoftWare Radio - SWR) [6] comme une architecture utilisant un matériel générique qui peut être programmé ou reconfiguré en "software" [2], afin de supporter de nombreuses normes de transmission sur une plateforme unique. Cela a été considéré comme une façon d'apporter de la flexibilité en pouvant ainsi traiter un grand nombre de signaux et services à coût et complexité raisonnables. Ces équipements pouvant supporter plusieurs normes seront alors qualifiés de "multi-standards". Une des idées clés du concept de SWR est d'exploiter les opérations de traitement communs entre les différentes normes qui sont destinées à être mises en œuvre sur une plate-forme commune. Dans ce contexte, la technique appelée *la paramétrisation* a été introduite [35, 36]. Cette approche permet à plusieurs normes d' utiliser les mêmes éléments de traitement et de partager leurs coûts.
Deux approches ont été dégagées dans le contexte de la paramétrisation, *l'approche théorique* et *l'approche pragmatique*. La première approche, sur laquelle ce travail de cette thèse se base, permet de trouver des éléments communs en utilisant une structure graphique d'un système multi-standards suggéré dans [49, 50] pour aboutir à des équipements optimisés et mutualisés. En ce qui concerne l'approche pragmatique, elle identifie les opérateurs communs en fusionnant des architectures proches les unes des autres, comme cela a été fait pour créer le mode dual de l'opérateur FFT dans [46] et les opérateurs communs basés sur Linear Feedback Shift Register (LFSR) dans [47] et [48].

Une approche graphique pour concevoir un équipement multi-standards flexible ("flexible" dans le sens où il pourra basculer d'une norme à une autre de façon transparente pour l'utilisateur) est proposée dans [50]. Cette approche décrit, selon un diagramme, les relations entre les différents blocs de traitement du système, en utilisant une décomposition par niveaux de granularité. Cette approche s'inscrit dans la technique théorique de la paramétrisation. Il s'agit d'un modèle mathématique dans lequel il est nécessaire d'effectuer les ajustements de granularité nécessaires, en résolvant un problème

d'optimisation associé au digramme. Des approches stochastiques donnant des solutions proches de l'optimum ont été proposées dans [60] pour résoudre ce problème, plutôt que d'utiliser des techniques déterministes, ce problème d'optimisation ayant été intuitivement considéré comme trop complexe. Puisque le problème a été formulé sous forme d'un diagramme dans [50], notre objectif est d'abord de le modéliser théoriquement en utilisant la théorie des graphes. Cette modélisation va ensuite nous permettre d'étudier et de créer un nouvel outil d'optimisation pour résoudre ce problème.

Le diagramme caractérisant les appels de fonctions à fonctions d' un équipement multi-standards permet de mettre à jour toutes les possibilités de réalisation possibles. Parmi toutes ces possibilités, seules quelques unes répondent à des critères précis associés dans notre travail au coût et au temps d'exécution. Une fonction de coût, exprimant les contraintes vis à vis de ces critères doit donc être minimisée permettant d'identifier les configurations optimales et ainsi les opérateurs communs entre standards.
Notre objectif est donc de choisir la configuration permettant d'optimiser le coût imposé par la fonction (de coût) suggérée dans [49]. Dans ce cas, il est dans un premier nécessaire d'étudier la complexité de ce problème d'optimisation afin de sélectionner les outils de la théorie des graphes permettant de résoudre ce problème. Une fois la technique d'optimisation appropriée proposée, une configuration optimale d'un système multi-standards peut être suggérée.

L'articulation du travail est alors la suivante:
Le chapitre 1 présente les principes de la radio logicielle et les défis qui y sont associés, ce qui entraîne l'émergence de sa version pratique appelée la radio logicielle restreinte (Software-Defined Radio-SDR). Les deux approches de la paramétrisation sont ensuite introduites, en insistant plus particulièrement sur l'approche théorique. Dans cette dernière technique, l'approche graphique d'un système multi-standards est détaillée ainsi que la fonction de coût proposée [49, 50].

Le chapitre 2 présente en détails deux théories nécessaires pour notre travail : la théorie des graphes et la théorie de la complexité.

Le chapitre 3 fournit un modèle théorique du problème lié à la conception d'un système multi-standards, en utilisant la théorie des graphes et plus précisément les hypergraphes orientés. Ensuite, dans ce chapitre, une étude de complexité est menée afin d'estimer le nombre total de possibilités engendrées par la conception d'un système multi-standards. On montre que ce problème est complexe sous une condition particulière et bien identifiée.

Dans le chapitre 4, l'étude se porte sur les différentes options de mise en œuvre et on montre que certaines d'entre elles peuvent être ignorées en résolvant notre problème d'optimisation. Cela a permis de proposer un nouvel algorithme (en utilisant des notions de modélisation différentes liées aux hypergraphes orientés) dont la complexité de calcul est ensuite détaillée. Après avoir développé un code pour notre algorithme en langage C, on a utilisé cet algorithme pour résoudre notre problème d'optimisation sur plusieurs exemples, afin d'etablir ses performances. La différence entre cet algorithme et ceux précédemment proposés pour ce problème est aussi soulignée. Au contraire des techniques heuristiques,

l'algorithme proposé est capable de fournir la solution exacte (et non approchée) associée au coût minimal imposé par la fonction de coût.

0.1 Techniques de paramétrisation pour la radio logicielle restreinte

0.1.1 Emergence de la radio logicielle restreinte

Les concepteurs des radiocommunications concentrent aujourd'hui leur attention sur le développement d' équipements flexibles multi-standards (capables de supporter plusieurs normes) s'affranchissant autant que possible des normes de télécommunications afin de faire face à l'innovation technologique toujours croissante. La radio logicielle regoupe un ensemble de techniques visant à répondre à ces évolutions.

Un émetteur-récepteur sera considéré comme "radio logicielle" si ses fonctions de communication sont réalisées sous forme de programmes exécutés sur un processeur approprié. La radio logicielle idéale a pour principe de numériser le signal reçu directement au niveau de la sortie de l'antenne. Cependant, de nombreux obstacles ne permettent pas aujourd'hui d'envisager une telle approche surtout au niveau de la partie analogique [1, 4] et des convertisseurs de traitement numérique [25, 26]. Ainsi, une version pratique de la radio logicielle, appelée la radio logicielle restreinte (Software-Defined Radio (SDR)) a été présentée dans [3]. Elle est basée sur le principe d' une numérisation en fréquence intermédiaire (IF). Cette fréquence intermédiaire (IF) est plus faible que la fréquence radio (RF).

Pour réaliser un équipement supportant plusieurs normes, une approche naturelle consiste à juxtaposer les chaînes de traitement dédiées à chaque standard. C'est ce que l'on appelle l'approche Velcro [41], où le passage d'une norme à l'autre (ou d'un service à l'autre) s'effectue par un simple switch. Cependant, cette approche n'est pas optimale car non évolutive du fait de l'augmentation progressive des normes supportées que l'équipement devra être amené à supporter. De plus, elle n'exploite pas du tout les aspects communs entre les standards qui peuvent permettre de gagner en complexité de réalisation. Cette dernière approche s'appelle la paramétrisation [35], développée dans la partie suivante.

0.1.2 Les techniques de paramétrisation

Le principe de la paramétrisation est de rechercher les caractères communs entre les traitements de différents standards puis d'en proposer des architectures communes et flexibles. Il peut être considéré comme un processus d'optimisation préliminaire incontournable pour concevoir des systèmes multi-standards flexibles. Cette approche vise à concevoir des systèmes multi-standards basés sur un jeu d'opérateurs opérateurs (ou fonctions) dans lesquels leur exécution est pilotée par un simple passage de paramètres, d'où le nom de paramétrisation [37].

La paramétrisation se décline selon deux approches: l'approche théorique et l'approche pragmatique. L'approche théorique consiste à lister de façon hiérarchique tous les appels de fonctions possibles dans un terminal pouvant supporter plusieurs normes. Ensuite, un processus d'optimisation est lancé afin d'identifier les niveaux de granularité optimaux (selon

une fonction de coût), à partir desquels des composants peuvent être considérés comme des «blocs communs de communication» permettant leurs réutilisations par plusieurs applications. Les opérateurs ainsi identifiés seront alors considérés comme communs (CO pour Common Operators). En revanche, l'approche pragmatique est une approche plus réaliste et pratique conçue pour identifier ou créer des opérateurs communs possibles à partir des briques de bases déjà existantes. L'approche pragmatique, contrairement à l'approche théorique adopte la démarche suivante : on essaie d'adapter un traitement donné (initialement prévu pour une fonction ou un standard donné) à d'autres traitements de façon à les rendre communs.

0.1.2.1 L'approche pragmatique de la paramétrisation

L'approche pragmatique consiste dans un premier temps à identifier dans la littérature les traitements proches (tant au niveau algorithmique qu'architectural), puis dans un second temps à réaliser un opérateur générique qui devra alors être reconfigurable. A titre d'exemple, une architecture reconfigurable d'une transformée de Fourier rapide (FFT) a été crée dans [46]. Un autre opérateur a été proposé dans [51] mutualisant deux algorithmes largement utilisés dans les systèmes de communication sans fil: ceux de Viterbi et de FFT. Leurs similitudes fonctionnelles et architecturales ont été décrites dans [51].

0.1.2.2 L'approche théorique de la paramétrisation

C'est cette approche qui a été privilégiée dans ce travail. Dans ce contexte, l'ensemble des fonctions réalisées par un terminal multi-standards est représenté par un diagramme [49, 50] dont chaque sommet représente un élément de traitement (Processing Element - PE) fonctionnel qui occupe un niveau de granularité donné . Chacun de ces éléments de traitement peut soit être directement mis en œuvre dans le système, ou représente une fonctionnalité obtenue en appelant des éléments de traitement de niveaux inférieurs. Deux dépendances de sommets, le "OU" et le "ET", ont été nécessaires pour illustrer clairement les besoins d'implémentation de chaque bloc et pour décrire le type de connexions entre les blocs de niveaux supérieurs et ceux de niveaux inférieurs.
Une dépendance de type "OU" (partie gauche de la figure 1) signifie qu'un seul des sommets descendants, appelé plusieurs fois, est nécessaire pour mettre en œuvre le sommet parent ((B ou C sont chacun capable de mettre en œuvre A). Une dépendance de type "ET" (partie droite de la figure 1) signifie que tous les sommets descendants sont nécessaires pour mettre en œuvre le sommet parent (B et C sont nécessaires tous les deux pour mettre en œuvre le bloc A, B et C étant "appelés" indifféremment pour réaliser A).
Notons que le même processus peut être effectué par le bloc A de niveau n, au lieu d'utiliser B ou C (partie gauche de la figure 1) ou de mettre en œuvre B et C à la fois (partie droite de la figure 1) qui sont des blocs de niveaux inférieurs. Cela se fera avec des temps d'exécution plus faibles mais avec un coût beaucoup plus élevé. Par contre, le fait d'utiliser les blocs de niveau $n-1$ va diminuer le coût, mais augmenter le temps d'exécution du système. Un compromis entre temps d'exécution et complexité est donc à trouver et c'est tout l'objectif des études liées à la paramétrisation.

En utilisant les deux dépendances "ET" et "OU", on peut illustrer une structure graphique d'un système multistandards comme celui de la figure 1.6 supportant deux normes dif-

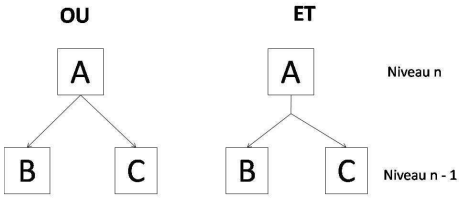

Figure 1: "OU" et "ET" dépendance

férentes (Wifi et UMTS). Il représente à titre d'illustration une version simplifiée du système complet (le graphe complet d'un système mutli-standard serait énorme à réaliser. Cela n'est pas l'objet de cette thèse de développer les graphes complets). Dans tous nos travaux, nous allons simplement présenter des illustrations graphiques réalisables ou quelques figures fournies par d'autres études antérieures.

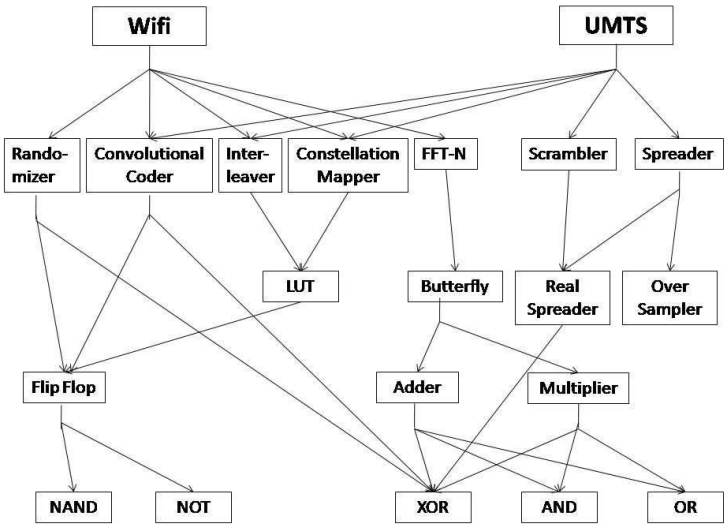

Figure 2: Structure simplifiée d'un système multi-standards (supportant Wifi et UMTS)

Cette approche fournit au concepteur toutes les options possibles afin de sélectionner un ensemble d'opérateurs pour réaliser une fonctionnalité donnée. Notre objectif est d'aider le concepteur à déduire un jeu d'opérateurs communs (COs) les plus adaptés d'un système multi-standards donné en recherchant le meilleur compromis entre flexibilité et efficacité et ceci en optimisant une fonction de coût donnée. .

La structure graphique du système multi-standards que l'on vient de présenter fournit toutes les options capables pour mettre en œuvre les différentes normes présentes dans le diagramme [50]. Une fonction de coût, une fonction, qui calcule le coût de ces options, a été proposée [49]. Cette fonction de coût, lorsqu'elle est optimisée, aidera à choisir

l'option ayant le coût minimal et de sélectionner ensuite les opérateurs communs associés. Les paramètres utilisés dans la fonction de coût incluent à la fois la notion de flexibilité et d'efficacité de l'implémentation.

Il a été associé à chaque élément de traitement du système, un coût de fabrication (*Building Cost - BC*) et un coût de calcul (*Computational Cost - CC*), et sur chaque arc du diagrame le nombre d'appels (*Number of Calls - NoC*). Dans le détail :

- **Le coût de fabrication** correspond au coût d'un élément de traitement capable d'exécuter une fonction donnée. Ce coût est "payé" juste une fois, indépendamment du nombre de fois où cet élément de traitement sera appelé. La réutilisation des éléments de traitement fournit ainsi une idée sur la réduction de coût.

- **Le coût de calcul** est associé au temps pris par un élément de traitement pour calculer une fonction donnée. Ce coût doit être calculé à chaque fois qu'un élément de traitement est appelé par ceux de niveaux supérieurs.

- **Le nombre d'appels** représente le nombre nécessaire de fois que des blocs de niveaux inférieurs seront appelés par leur sommet parent. Il s'agit d'un facteur multiplicatif qui sera associé aux arcs.

Dans la suite, nous allons décrire le développement de la fonction de coût proposé. Sur la base des paramètres introduits précedemment (BC et CC), notre objectif est de minimiser le coût total de fabrication du système ainsi que son coût total de calcul (en terme de temps d'exécution). Dans ce contexte, notre fonction de coût sera une fonction à deux objectifs : réduction du coût de fabrication d'un terminal multi-standards en proposant des opérateurs communs, tout en assurant des temps d'exécution raisonnables. Ces deux objectifs sont souvent en opposition : en effet la réduction du coût de fabrication augmentera le coût total de calcul et vice versa. Remarquons que plus un opérateur sera commun (à plusieurs standars) et plus il sera amené à être exécuté souvent. Plus un opérateur sera commun et plus il devra se "partager" entre un grand nombre de traitements, ce qui complexifie le problème du cadencement des taches (non abordé dans cette thèse).

Minimiser le coût de fabrication s'écrit alors :

$$min \sum_i BC_i.N_i \tag{1}$$

où $\sum_i BC_i.N_i$ est le coût total de fabrication de tous les éléments de traitement qui sont présents dans l'équipement multi-standards. $N_i \in \{0, 1\}$ indique si le i^{th} sommet est présent ou pas dans l'équipement.

Maintenant, considérons le second objectif qui consiste à minimiser le coût de calcul. Il peut s'écrire ainsi:

$$min \sum_{n} \sum_{k} CC_k((S_n)_{n\in\{1,2\cdots,N\}}), \tag{2}$$

où

- $\sum_{k} CC_k((S_n)_{n\in\{1,2,\cdots,N\}})$ représente le coût de calcul total pour chaque standard S_n.
- $\sum_{n}\sum_{k} CC_k((S_n)_{n\in\{1,2,\cdots,N\}})$ est le coût de calcul total pour tous les N standards S_n.

Ce problème est un problème d'optimisation multi-objectifs. Les fonctions "objectif" de ce problème sont incommensurables comme la plupart des problèmes d'optimisation d'ingénierie. L'approche "somme pondérée" [58, 59], a été considérée dans [60] pour définir la fonction de coût en raison de sa simplicité. Elle consiste à regrouper les fonctions d'optimisation différentes dans une seule fonction, accompagnée des coefficients de poids associés à chaque fonction objectif. Ainsi, la fonction de coût bi-objectifs qui combine la fonction à deux objectifs incommensurables des équations 1.1 et 1.2 aura la forme :

$$Cout = (\overline{w} \sum_{i} BC_i.N_i + \sum_{n}\sum_{k} w_n CC_k((S_n)_{n\in\{1,2\cdots,N\}})), \tag{3}$$

où $\overline{w}$ est le poids octroyé au coût de fabrication de l'équipement, et w_n est le poids d'un seul standard S_n. En effet, un seul standard doit pouvoir s'exécuter à un instant donné. Notre contribution doit aider à concevoir un terminal multi-standards, en minimisant cette fonction de coût.
La résolution de ce problème d'optimisation apportera alors une solution assurant le compromis entre la complexité et l'efficacité. Bien que cette fonction de coût semble évidente, le problème n'en reste pas moins très difficile. Dans notre travail, les poids $\overline{w}$ et w_n dans la fonction de coût de l'equation 3 seront neutralizés à 0.5 sauf si le contraire est indiqué, car ils peuvent varier en fonction des préoccupations de chaque concepteur.
L'intérêt de réduire la consommation d'énergie, le coût et la taille du système radio tout en atteignant les performances attendues et en profitant des améliorations technologiques, a été une motivation pour rechercher des aspects d'optimisation afin de concevoir des systèmes de radiocommunications flexibles. L'objectif de cette thèse est de se concentrer sur les aspects de la radio logicielle globale et de trouver de nouvelles techniques d'optimisation théoriques des équipement de la radio logicielle restreinte flexibles, en utilisant la *théorie des graphes*. Ainsi, dans le chapitre suivant, nous allons présenter des définitions fondamentales de la théorie des graphes ainsi que quelques définitions de base liées à la théorie de la complexité, qui seront ensuite utilisées pour étudier la complexité de notre problème d'optimisation.

0.2 La théorie des graphes et ses applications

0.2.1 Définitions nécessaires de la théorie des graphes

Ce paragraphe présente quelques définitions nécessaires à la compréhension de la suite du document.

Graphe Un graphe G est une paire ordonnée $(V(G), E(G))$ comprenant l'ensemble fini, non-vide $V(G)$ dont les éléments sont appelés *des sommets* et l'ensemble $E(G)$ de paires non ordonnées de sommets $V(G)$ (pas nécessairement distincts), appelées des arêtes. Une arête avec des extrémités identiques est appelée *une boucle*. Deux arêtes ou plus ayant la même paire d'extrémités sont appelés *des arêtes multiples*. Un *multigraphe* G est un graphe qui admet des arêtes multiples. Un graphe G comportant des arêtes multiples et des boucles est appelé un *pseudographe*. A l'opposé, un *graphe simple* n'admets ni boucles, ni arêtes multiples. [62].

Graphe orienté Un graphe orienté D est défini par une paire ordonnée d'ensembles $(V(D), A(D))$, où $V(D)$ est un ensemble fini, nonvide de sommets et $A(D)$ est un ensemble des paires ordonnées des sommets de $V(D)$ (pas nécessairement distincts) appelés des arcs. D'une façon analogue aux graphes, on peut définir *des multigraphes orientés*, *des pseudographes orientés* et *des graphes orientés simples*.

Hypergraphe Un hypergraphe [68] est une généralisation d'un graphe où une arête peut être connectée à n'importe quel nombre de sommets. Formellement, un hypergraphe H est défini par une paire $(V(H), E(H))$, où $V(H)$ est un ensemble de sommets nonvide et $E(H)$ est un ensemble fini de sous-ensembles de $V(H)$ (pas nécessairement nonvide) nommé *des hyperarêtes*.

Soit $H = (V(H), E(H))$ un hypergraphe et $x \in V(H)$.

Sous-hypergraphe Un *sous-hypergraphe* de H est un hypergraphe $H' = (X', D')$ tel que $X' \subseteq V(H)$ et $D' \subseteq E(H)$.

Etoile de centre x : l'étoile $S(x)$ de centre x est l'ensemble des hyperarêtes contenant x, c-à-d $S(x) = \{E \in E(H), x \in E\}$.

Degré d'un sommet Le *degré* de x dans H, noté $d_H(x)$, est le nombre d'hyperarêtes dans $E(H)$ comprenant x, c-à-d $d_H(x) = |S(x)|$.

Hypergraphe orienté C'est une généralisation d'un graphe orienté. Un hypergraphe orienté est un hypergraphe avec des hyperarêtes orientées (aussi appelées des hyperarcs) où un hyperarc E est une paire ordonnée $E = (X, Y)$ de sous-ensembles disjoints de sommets; X est l'ensemble d'origines de E notée $T(E)$ et Y est son ensemble d'extrémités notée $H(E)$ [69]. La figure 3 donne un exemple d'un hypergraphe orienté $H = (V, E)$ tel que:

$X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$

$E = \{E_1, E_2, E_3, E_4, E_5, E_6\}$ où:
$E_1 = (\{x_1, x_2\}, \{x_3\})$
$E_2 = (\{x_3\}, \{x_7\})$
$E_3 = (\{x_3, x_4\}, \{x_5, x_6\})$
$E_4 = (\{x_7\}, \{x_1\})$
$E_5 = (\{x_7\}, \{x_8\})$

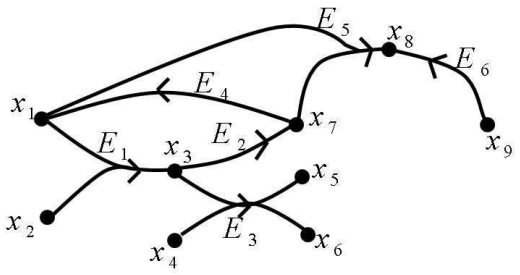

Figure 3: Un hypergraphe orienté $H = (X, E)$

$E_6 = (\{x_9\}, \{x_8\})$

Soit $H = (V(H), E(H))$ un hypergraphe orienté et $v \in V(H)$.

Sous-hypergraphe orienté Un hypergraphe orienté $H' = (V(H'), E(H'))$ est appelé un *sous-hypergraphe orienté* de H si $V(H') \subseteq V(H)$ et $E(H') \subseteq E(H)$.

B-hypergraphe et F-hypergraphe [69] Un *hyperarc retour*, abrégé par *B-arc*, est un hyperarc $E = (T(E), H(E))$ où $|H(E)| = 1$; un *hyperarc devant*, abrégé par *F-arc*, est un hyperarc E tel que $|T(E)| = 1$.
Un *B-hypergraphe* (ou simplement *B-graphe*) est un hypergraphe orienté dont ses hyperarcs sont tous des B-arcs; De même, un *F-hypergraphe* (ou simplement *F-graphe*) est un hypergraphe dont tous les hyperarcs sont F-arcs.

BF-réductions d'un hyperarc [69] Soit $E = (T(E), H(E))$ un hyperarc dans un hypergraphe orienté H. Une BF-réduction de E est un hyperarc $(\{x\}, \{y\})$ où $x \in T(E)$ et $y \in H(E)$. Par exemple, $(\{x_1\}, \{x_3\})$ et $(\{x_2\}, \{x_3\})$ sont des BF-réductions de l'hyperarc E_1 dans la figure 3.
Chaque BF-réduction de E est appelée un arc de E.

Etoile devant et Etoile retour [69] L'étoile devant (FS pour Forward Star) et l'étoile retour (BS pour Backward Star) du sommet v sont respectivement définies par:
$FS(v) = \{E \in E(H), v \in T(E)\}$ et $BS(v) = \{E \in E(H), v \in H(E)\}$.
Si $X = (V(X), E(X))$ est un sous-hypergraphe orienté de l'hypergraphe orienté H, alors l'étoile devant et retour dans X du sommet v ($v \in V(X)$) seront définies comme suit:
$FS_X(v) = \{E \in E(X); v \in T(E)\}$ et $BS_X(v) = \{E \in E(X); v \in H(E)\}$.

Chemins orienté dans les hypergraphes orientés [69] Dans un hypergraphe orienté H, un chemin de r à n $(r, n \in V(H))$ est défini par une séquence de sommets et d' hyperarcs $P = (v_1 = r, E_{i_1}, v_2, E_{i_2}, v_3, \cdots, E_{i_q}, v_{q+1} = n)$ où:
$r \in T(E_{i_1})$, $n \in H(E_{i_q})$ et $v_j \in H(E_{i_{(j-1)}}) \cap T(E_{i_j})$ $j = 2, \cdots, q$.

Ce chemin est appelé un rn-chemin, où r est son origine et n son extrémité. On désigne $V(P)$ et $E(P)$ par l'ensemble des sommets et d'hyperarcs traversés via P et $l(P)$ par le nombre des hyperarcs dans le chemin, appelé la longueur

du chemin. On peut donc écrire $V(P) = \{v_1, v_2, \cdots, v_{q+1}\}$, $E(P) = \{E_{i_1}, E_{i_2}, \cdots, E_{i_q}\}$ et $l(P) = q$. Par exemple, dans l'hypergraphe orienté de la figure 3, on a $Q = (x_2, E_1, x_3, E_2, x_7, E_5, x_8)$ un chemin orienté de x_2 à x_8 de longueur 3, où $V(Q) = \{x_2, x_3, x_7, x_8\}$ et $E(Q) = \{E_1, E_2, E_5\}$.

Degré d'un sommet Le *degré intérieur de* v, noté par $d_H^-(v)$, représente le nombre d'hyperarcs qui contient v dans leur ensemble d'extrémités, tandis que le *degré extérieur de* v est le nombre d'hyperarcs contenant v dans leur ensemble d'origines, noté par $d_H^+(v)$. Par conséquent, on peut écrire: $|BS(v)| = d_H^-(v)$ et $|FS(v)| = d_H^+(v)$.

Théorème 1. *Pour un hypergraphe orienté* $H = (V(H), E(H))$*, on a* $\sum_{x \in V(H)} d_H^+(x) = \sum_{E \in E(H)} |T(E)|$.
De même, on a $\sum_{x \in V(H)} d_H^-(x) = \sum_{E \in E(H)} |H(E)|$.

0.2.2 La théorie de la complexité

Un problème de décision Π est un problème qui n'a que deux solutions possibles, soit la réponse «oui» soit la réponse «non». De nombreux problèmes qui se posent dans la pratique sont des problèmes d'optimisation plutôt que des problèmes de décision. Néanmoins, tout problème d'optimisation (de minimisation ou de maximisation) peut être transformé en un problème de décision en se fixant un borne $B \geq 0$, la question suivante est alors formulée : Peut-on trouver une solution qui satisfasse la condition du problème, mais dont le coût ne dépasse pas B (dans le cas d'un problème de minimisation)?

0.2.2.1 La classe P

Un algorithme a une définition formelle basée sur la *Machine de Turing*, qui est utilisé comme un modèle de calcul par l'ordinateur en raison de sa simplicité.

De manière informelle, la complexité d'un algorithme est une fonction C qui représente le nombre maximum d'étapes de calcul de base nécessaires pour son exécution (comme les opérations arithmétiques et les comparaisons), prise sur toutes les variances[(1)] du problème possibles dont la taille est s (C est une fonction de s). C'est ce que l'on appelle *la complexité du pire des cas*. On dira qu'une fonction f a un ordre ne dépassant pas celui de g (notation $f = O(g)$), s'il existe deux constantes k et n_0 tel que $f(n) \leq kg(n)$, $\forall\ n \geq n_0$. Un algorithme est dit *polynomial* (resp. exponentiel) si sa fonction de complexité C est $O(f)$ avec f une fonction polynomiale (resp. *exponentielle*). La classe P regroupe l'ensemble de tous les problèmes qui peuvent être résolus dans un temps polynomial. Formellement, la classe P est un ensemble qui comprend tous les problèmes dont les algorithmes sont réalisés par un programme sur la machine de Turing déterministe de temps polynomial.

[(1)] une variance du problème est aussi appelée une instance du problème (In English: instance of a problem)

Un problème pour lequel aucun algorithme n'est trouvé pour le résoudre dans un temps polynomial est considéré comme un problème difficile.

0.2.2.2 La classe NP

Un *algorithme non déterministe* est un algorithme qui comporte deux étapes, nommé l'étape *de recherche* et l'étape *de vérification*. Etant donnée une variance I d'un problème, la première étape "recherche" une solution S. Ensuite I et S sont les données d'entrée de l'étape de vérification qui calcule de manière déterministe classique, en renvoyant soit la réponse «oui», soit la réponse «non», soit en calculant sans jamais s'arrêter. Un algorithme non déterministe est dit *polynomial* si pour toutes les variances I de taille s dont la réponse est «oui», on peut trouver une solution qui termine l'étape de vérification après un nombre d'étapes de calcul de base qui est une fonction polynomiale de s.

La classe NP est définie par l'ensemble des problèmes de décision qui peuvent être résolus par un *algorithme non-déterministe polynomial*. Pour résumer, un problème de décision est un problème NP si pour une variance I dont la réponse est «oui», et après avoir identifié une certaine solution S, on peut vérifier que la réponse pour I et S est «oui» en temps polynomial. Formellement, la classe NP est définie par l'ensemble des problèmes de décision dont les programmes sont réalisés par une machine de Turing non-déterministe de temps polynomial.

L'une des questions ouvertes les plus fondamentales dans toutes les mathématiques est la conjecture de Cook-Edmond-Levin: P $\neq$ NP. Si P est différent de NP, alors la distinction entre P et NP-P est importante. Dans ce contexte, nous allons fournir dans la suite de ce chapitre une caractérisation des problèmes les plus difficiles dans NP-P, appelés problèmes NP-complet.

0.2.2.3 Problèmes NP-complet

Transformation polynomiale Une *transformation polynomiale* d'un problème de décision Π_1 en un problème de décision Π_2, noté $\Pi_1 \propto \Pi_2$, est une fonction $f : D_{\Pi_1} \rightarrow D_{\Pi_2}$ qui satisfait les deux conditions suivantes:

1. f est calculable par un algorithme à temps polynomial.
2. f transforme une variance I de Π_1 en une variance $f(I)$ de Π_2 tel que la réponse pour I par rapport à Π_1 est *«oui»* si et seulement si la réponse pour $f(I)$ par rapport à Π_2 est *«oui»*.

Lemme 1. *Si* $\Pi_1 \propto \Pi_2$*, alors* $\Pi_2 \in P$ *implique* $\Pi_1 \in P$ *[63, 61].*

Notez que la relation «transformabilité polynomiale "$\propto$" est réflexive. En effet, elle est particulièrement utile car il a été prouvé que c'est une relation transitive [63, 61]. Cela signifie que si $\Pi_1 \propto \Pi_2$ et $\Pi_2 \propto \Pi_3$, alors $\Pi_1 \propto \Pi_3$.

Définition 1. Un problème de décision Π est dit NP-complets si:

- $\Pi \in$ NP.
- $\forall\ \Pi' \in$ NP, $\Pi' \propto \Pi$.

Théorème 2. Π *est NP-complet (NPC) s'il satisfait :*

1. $\Pi \in$ *NP.*
2. $\exists\ \Pi' \in$ *NPC, tel que* $\Pi' \propto \Pi$.

Notez qu'un problème de décision est dit NP-dur s'il ne satisfait que la deuxième propriété, c'est à dire qu'il diffère des problèmes NP-complets par le seul fait qu'il n'a pas besoin d'être un problème NP.

Dans ce cas, on a encore besoin d'un premier problème NP-complet. Ce premier problème est celui de "Satisfaisabilité" (voir [101, 63])).

Les problèmes NP-complet sont considérés difficiles tant qu'aucun algorithme de temps polynomial n'a pas été trouvé pour résoudre l'un d'eux. La plupart de ces problèmes sont généralement résolus par des méthodes approximatives, appelées *des heuristiques*. Une classe particulièrement simple et naturelle de ces heuristiques est la classe d'algorithmes gloutons [72].

0.2.3 Applications de la théorie des graphes

Presque toutes les questions sur les graphes (ou graphes orientés) nécessitent une étude de chaque arête (et dans le processus chaque sommet) au moins une fois. Les approches DFS (Depth-first search) et BFS (Breadth-First search) cherchent respectivement les arêtes d'un graphe quand ils se déplacent d'un sommet à un autre. Les deux techniques de recherche s'exécutent en temps polynomial.

- méthodologie BFS : une fois au sommet v de G, il examine toutes les arêtes (ou arcs) avec l'extrémité v, puis passe à un autre sommet de G qui est adjacent à v.
- méthodologie DFS: une fois au sommet v de G, il examine une seule arête (ou arc) avec l'extrémité v, puis passe à l'autre extrémité de cette arête (ou arc).

L'approche BFS est l'archétype de nombreux algorithmes de graphes importants, y compris celui de Prim [75] pour trouver un arbre couvrant de poids minimal (Minimum Spanning Tree - MST) et l'algorithme de Dijkstra [78, 62] pour chercher les plus court chemins d'un sommet unique (single-source-shortest-path problem) etc $\cdots$. DFS semble également être une approche utile, pour vérifier par exemple la connectivité dans les graphes, isomorphisme, etc $\cdots$. (voir [61] pour de plus amples renseignements).

La théorie des graphes a été utilisée pour résoudre d'autres problèmes d'optimisation, comme celui du voyageur de commerce (Traveling Salesman Problem - TSP) [71] qui cherche à optimiser la distance à parcourir pour visiter tous ses clients. Ce problème a été modélisé par un graphe pondéré et s'avère être un problème NP-complet [63]. Plusieurs méthodes heuristiques ont été proposées, donnant une route très proche

de la plus courte, mais ne la garantissent pas ([73, 82]).

Diverses problématiques du domaine des télécommunications et des réseaux ont été formulés soient selon des modèles de graphes théoriques, soient des hypergraphes, des graphes orientés ou des hypergraphes orientés. On peut mentionner le problème d'affectation de canal (channel assignment problem) décrit comme suit : une zone de couverture est divisée en cellules autour de chaque émetteur. Les canaux sont assignés aux différentes cellules, dans lesquelles le même canal peut être utilisé simultanément par plusieurs cellules, dans la mesure où elles sont suffisamment séparées pour éviter les interférences. L'objectif est de trouver une affectation des canaux aux différentes cellules telle que l'interférence correspondant est acceptable (en dessous d'un niveau donné), tout en utilisant le nombre minimum de canaux. Ce problème est un problème NP-dur [83], parce que l'un de ses cas particuliers est formulé selon le problème de coloration des graphes qui est connu pour être NP-dur [63]. Un algorithme exponentiel général a été proposé pour ce problème dans [84]. D'autres problèmes comme des problèmes de réseaux ont été modélisés par des hypergraphes [64] et des hypergraphes orientés [65].

0.3 Apport de la théorie des graphes dans l'approche théorique de la paramétrisation

Dans le premier chapitre, nous avons introduit une approche graphique fournissant une illustration de toutes les options pour réaliser un équipement supportant plusieurs normes (dit multi-standards). Nous avons aussi présenté une formulation de la fonction qui calcule le coût de chaque alternative de mise en œuvre sélectionnée. Dans ce chapitre, on va modéliser tous ces aspects et d'autres théoriquement en utilisant la théorie des graphes ou plus précisément, en utilisant les hypergraphes orientés. Ceci constituera la première partie de ce chapitre. Dans la deuxième partie, on menera une étude sur le nombre total d' options capables de mettre en œuvre un système multi-standards, ce qui donnera une idée très précise sur la difficulté du problème posé. Ensuite dans la troisième partie, on étudiera la complexité de notre problème d'optimisation.

0.3.1 Un modèle formel pour les différents aspects d'un équipement multi-standards

0.3.1.1 Un modèle mathématique de la structure graphique d'un équipement multi-standards

La figure 4 fournit l' exemple d'un système à deux standards (notés S et T). Dans cette figure, des valeurs numériques sont associés aux sommets qui représentent le BC/CC du noeud, et d'autres associés aux arcs représentant le nombre d'appels des blocs (NoC).

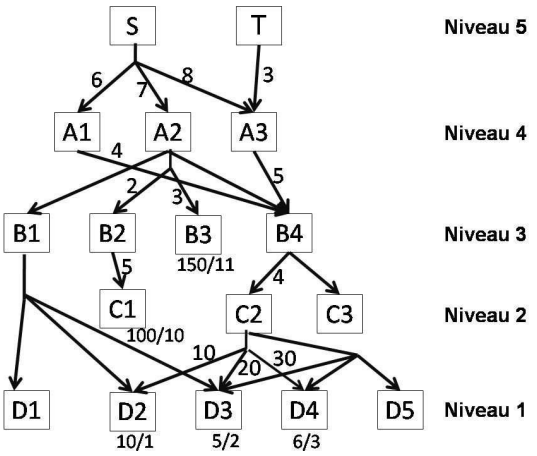

Figure 4: Structure globale d'un système multi-standards illustrant la décomposition des standards S et T sur 4 niveaux

La structure graphique de l'approche théorique de la paramétrisation présentée dans le chapitre 1 peut être représentée de manière formelle. En effet, la structure graphique du système multi-standards peut être décrite comme un hypergraphe orienté, indispensable pour représenter les deux dépendances "OU" et "ET". Cet hypergraphe orienté est défini par le couple (V, E), où l'ensemble de sommets V comprend les blocs (fonctions et opérateurs) présents dans la figure, et où l'hyperarc $e \in E$ contient le sommet parent comme un sommet d'origine; quant aux sommets descendants capables d'effectuer la tâche du bloc parent, ils constituent l'ensemble d'extrémités de e.
A chaque fois que l'on est confronté à une dépendance de type "ET", l'hyperarc sera formé de telle sorte que le sommet parent soit le sommet d'origine, et tous les sommets descendants via ce "ET" forment les sommets dans l'ensemble d'extrémités de l'hyperarc. Par contre, lorsque l'on est confronté à une dépendance de type "OU", le sommet parent forme le sommet d'origine de l'hyperarc tandis que le seul sommet descendant nécessaire via cette dépendance "OU" forme le sommet d'extrémité de l'hyperarc.

Par exemple, $(\{S\}, \{A1, A2, A3\})$, $(\{B4\}, \{C2\})$, $(\{B4\}, \{C3\})$,
$(\{A2\}, \{B1\})$, $(\{A2\}, \{B2, B3\})$, $\cdots$ etc appartiennent à l'ensemble E d'hyperarcs de l'hypergraphe orienté dans la figure 4.
Remarquons que cette représentation du système multi-standards par un hypergraphe orienté est en fait un F-graphe puisque chaque hyperarc ne contient qu'un seul sommet d'origine qui est le sommet parent.

Chaque élément de traitement inclu dans la structure graphique d'un système multi-standards (représenté par un F-graph H) occupe un certain niveau. Ce qui suit est une remarque sur le niveau attribué à chaque élément de traitement. Supposons que l'on décompose les normes prises en charge sur $n-1$ niveaux différents (donc au

total, on aura n niveaux, y compris le plus haut niveau des normes). Le niveau d'un bloc v dans la représentation d'un système multi-standards comme un hypergraphe orienté, noté $L(v)$, est défini par:

$$L(v) = n - \max_{x/d_H^-(x)=0} (\max_{P\,:\,xC-chemin} (l(P))), \tag{4}$$

où $l(P)$ représente la longueur du chemin P.
Le niveau de chaque bloc dans la figure 4 est identifié à droite de la figure.

Dans ce qui suit, on définit $\Im$ par l'ensemble des normes dans le système multi-standards qui occupent le plus haut niveau. Ainsi, $\Im = \{S, T\}$ dans le cas de la figure 4.

La structure graphique du système multi-standards nous informe sur toutes les options possibles pouvant mettre en œuvre sa conception. Avant d'expliquer comment chacune de ces options peut être représentée graphiquement, on va proposer quelques définitions nécessaires pour le reste de notre travail concernant les hypergraphes orientés.

Définition 2: Poids d'un chemin dans un hypergraphe orienté
Poids d'un chemin dans un hypergraphe orienté
Soit $P = P_{rn} = (v_1 = r, e_{i_1}, v_2, e_{i_2}, v_3, \cdots, e_{i_q}, v_{q+1} = n)$ un rn-chemin et $e_{i_j} \in E(P)$. On définit la BF-réduction via le chemin P de e_{i_j} par sa BF-réduction particulière obtenue en sélectionnant le sommet précédant de e_{i_j} dans le chemin P afin qu'il soit le sommet queue et le sommet suivant de e_{i_j} afin qu'il soit le sommet tête. On désigne par $BF_P(e_{i_j})$ la BF-réduction de e_{i_j} via P. Ainsi, selon cette définition, on obtient: $BF_P(e_{i_j}) = (\{v_j\}, \{v_{j+1}\})$ $j = 1, 2, \cdots, q$.

Supposons que l'on ait un hypergraphe orienté $H = (V(H), E(H))$ dans lequel un poids entier positif soit attribué à chaque arc de H. Pour tout chemin P entre r et n, on définit le poids de P par le produit des poids des BF-réductions via P de tous les hyperarcs dans $E(P)$. Ainsi, on peut écrire:

$$w(P) = \prod_{e_{i_j} \in E(P)} w(BF_P(e_{i_j})) \tag{5}$$

où $w(P)$ désigne le poids du chemin P et où $w(BF_P(e_{i_j}))$ représente le poids de $BF_P(e_{i_j})$ en H.

Par exemple, le poids du chemin allant de S à $C1$ dans la figure 4 (en passant par $A2$ et $B2$) est $w(\{S\}, \{A2\}) \times w(\{A2\}, \{B2\}) \times w(\{B2\}, \{C1\}) = 7 \times 2 \times 5 = 70$.

Définition 3. Ajout d'un hyperarc

Soit X un sous-hypergraphe orienté d'un hypergraphe orienté H tel que $E(X) \neq E(H)$ et soit e appartenant à $E(H)$ mais n'appartenant pas à $E(X)$. En ajoutant e à X, on obtient un sous-hypergraphe orienté $X^{'}$ de H, noté $X + e$, défini par :
$V(X^{'}) = V(X) \cup H(e) \cup T(e)$ et $E(X^{'}) = E(X) \cup \{e\}$.
$X + e$ est appelé sous-hypergraphe orienté de H induit par X et e.

Définition 4. un G-chemin

Soit H un hypergraphe orienté et $N \subseteq V(H)$.
On dit qu'un sous-hypergraphe orienté X est un G-chemin de H de racine N s'il satisfait:

1. $d_X^+(u) \in \{0, 1\} \quad \forall\, u \in V(X)$
2. $N \subseteq V(X)$
3. $\forall\, u \in V(X)$, il existe un chemin de v à u pour un sommet $v \in N$

0.3.1.2 Une représentation d'une option de mise en œuvre

On a illustré chaque option de mise en œuvre choisie par un hypergraphe orienté obtenu à partir de la structure graphique d'origine du système SDR multi-standards, appelé graphe généré (GNG). Il est défini de telle sorte que les opérateurs choisis dans la conception soient illustrés avec une étoile devant vide et on montre toutes les fonctions nécessaires que les blocs installés construisent, étape par étape, jusqu'à ce qu'ils atteignent les fonctionnalités des normes de plus haut niveau.

La figure 3.2 montre les graphes générés (obtenus à partir de la figure 4) de deux options différentes de mise en œuvre capables de réaliser S et T. Dans la première option, les opérateurs choisis sont $D2, D3, D4, C1, \&B3$ et le GNG correspondant est illustré sur la partie gauche de la figure 3.2. Quant à la deuxième option, les opérateurs choisis sont $D2, D3, D4, C1, B3\&B4$ et la représentation GNG de cette option est celle représentée sur la partie droite de la figure 3.2.
La seule différence entre ces deux options est que dans la première, les blocs $D2, D3, \&$ $D4$ sont utilisés pour mettre en œuvre les fonctionnalités de $A1\&A3$ passant à la fois par les blocs $C2\&B4$ alors que dans le deuxième choix, $D2, D3, \&D4$ sont utilisés pour réaliser bloc $A3$ mais le bloc $B4$ pour réaliser $A1$. Le cas de la deuxième option représente une alternative dans laquelle certains blocs de niveaux inférieurs sont installés dans la conception, ainsi que ceux de niveaux supérieurs qui peuvent être construits par ceux de niveaux inférieurs. (car les opérateurs $D2, D3$ et $D4$ seront installés dans la conception ainsi que $B4$ qui lui-même peut être réalisé par $D2, D3$ et $D4$).

Cependant, il faut noter que le GNG de la première option est un G-chemins de racine $\Im$, contrairement à la seconce option. Les graphes générés des options ressemblant à la seconde option ont toujours une partie dédoublée, ce qui contredit l'illustration

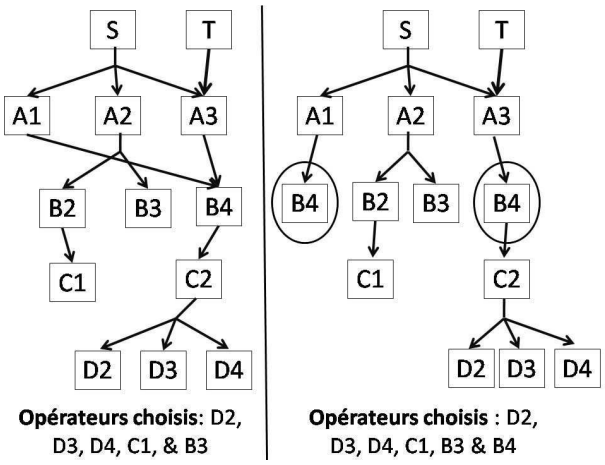

Figure 5: Les graphes générés (obtenus de la figure 4) de deux options de mise en œuvre différentes

d'un sous-hypergraphe orienté et donc ne correspondent pas à des G-chemins. Par conséquent, les options de mise en œuvre peuvent être divisées en celles dont les graphes générés sont des G-chemins de racine $\Im$, et en celles où l'on trouve une partie dédoublée dans leur graphes générés.

0.3.1.3 Description de l'hypergraphe orienté multi-standards à partir de l'hypergraphe mono-standard

Dans cette partie, on va expliquer théoriquement comment arriver à la structure graphique des alternatives capables de mettre en œuvre le terminal multi-standards, étant donné les structures graphiques de chacun des standards. Par exemple, la décomposition du Wifi et UMTS montrée dans la figure 1.6 est obtenue à partir des structures graphiques distinctes qui représentent les différentes alternatives à mettre en œuvre pour le Wifi et l'UMTS seuls.

Soit $T_1 = (V(T_1), E(T_1))$ un hypergraphe orienté représentant toutes les options de mise en œuvre pouvant implémenter un seul standard S_1 et $T_2 = (V(T_2), E(T_2))$ un hypergraphe orienté illustrant les différentes alternatives capables d'implémenter un autre standard S_2. On note T l'hypergraphe orienté illustrant les options de mise en œuvre pour les deux standards S_1 et S_2.

Il y aura fort probablement des opérateurs qui pourront être utilisés dans l'implémentation des deux standards. En conséquence $\exists V \subseteq V(T_1)$ & $\exists U \subseteq V(T_2)$ tel que :

$V = U$ au sens de l'opérateur.

Quand on dit "au sens de l'opérateur ", cela signifie en tenant compte des éléments de U & V comme des opérateurs fonctionnels tels que FFT, additionneurs , multiplieurs, etc.
Par exemple, si la FFT est un opérateur qui peut être utilisé dans les deux standards S_1 & S_2, elle pourrait être le sommet v_5 dans $V(T_1)$ et le sommet u_3 dans $V(T_2)$. On écrit alors $u_3 = v_5$ au sens de l'opérateur.

On définit la fonction $f : V \longrightarrow U$ tel que:
$\forall v \in V \quad f(v) = u \Leftrightarrow v = u$ au sens de l'opérateur .
Notons que f est une application bijective.

Définition 5. Soit $e \in E(T_2)$. Un hyperarc e' est défini par:

- $\forall\, r \in T(e) \cap U (resp.\ r \in H(e) \cap U),\ \ f^{-1}(r) \in T(e^{'})$ (resp. $f^{-1}(r) \in H(e^{'})$)
- $\forall\, r \in T(e) \backslash U$ (resp. $r \in H(e) \backslash U$), $\quad r \in T(e^{'})$ (resp. $r \in H(e^{'})$).

Dans l'hypergraphe orienté T, les opérateurs communs (entre $V(T_1)$ et $V(T_2)$) qui sont les éléments des ensembles U et V seront introduits juste une fois, et tous les hyperarcs dans $E(T_1)$ et $E(T_2)$ qui utilisent un opérateur commun donné le partagent. Par conséquent, l'hypergraphe orienté T sera défini par le couple $(V(T), E(T))$ tel que:

- $V(T) = V(T_1) \cup (V(T_2) \backslash U)$.
- $E(T) = E(T_1) \cup \{e^{'}\ / e \in E(T_2)\}$.

Une fois que T est donné, et étant donné un autre hypergraphe orienté T_3 qui illustre toutes les alternatives capables d'implémenter un autre standard S_3, on peut alors établir un hypergraphe orienté $T^{'}$ représentant les différentes alternatives capables de mettre en œuvre les 3 standards S_1, S_2 & S_3.
Il est alors évident que l'on peut définir un hypergraphe orienté pour un équipement multi-standards supportant n'importe quel nombre de standards.

0.3.1.4 L'équation formelle de la fonction de coût

Dans cette partie, on va établir une expression théorique alternative de la fonction de coût dans l'équation 3, qui évalue le coût de chaque option de mise en œuvre sélectionnée.

Dans la suite, on va noter $w(A, B)$ le nombre de fois que le bloc A appelle le bloc B. Considérons à nouveau le premier choix de mise en œuvre exposé dans la partie 3.1.2 (en choisissant les opérateurs $D2, D3, D4, C1, \& B3$) pour implementer les normes S et T de la figure 4. Le coût de la mise en œuvre via ce choix (selon l'équation 3) est

calculé comme suit:

Coût = $(((CC(D2) \times w(C2, D2) + CC(D3) \times w(C2, D3) + CC(D4) \times w(C2, D4)) \times w(B4, C2)) \times w(A3, B4)) \times w(S, A3) + (((CC(D2) \times w(C2, D2) + CC(D3) \times w(C2, D3) + CC(D4) \times w(C2, D4)) \times w(B4, C2)) \times w(A1, B4)) \times w(S, A1) + ((CC(C1) \times w(B2, C1)) \times w(A2, B2) + CC(B3) \times w(A2, B3)) \times w(S, A2) + (((CC(D2) \times w(C2, D2) + CC(D3) \times w(C2, D3) + CC(D4) \times w(C2, D4)) \times w(B4, C2)) \times w(A3, B4)) \times w(T, A3) + BC(D2) + BC(D3) + BC(D4) + BC(C1) + BC(B3)$

$= (((1 \times 10 + 2 \times 20 + 3 \times 30) \times 4) \times 5) \times 8 + (((1 \times 10 + 2 \times 20 + 3 \times 30) \times 4) \times 4) \times 6 + ((10 \times 5) \times 2 + 11 \times 3) \times 7 + (((1 \times 10 + 2 \times 20 + 3 \times 30) \times 4) \times 5) \times 3 + 10 + 5 + 6 + 100 + 150$

Après avoir développé l'équation ci-dessus et avoir effectué certaines factorisations, on obtient une expression générale de la forme :

$$Cout = \sum_{y/d^-_{GNG}(y)=0} \Big(\sum_{x/d^+_{GNG}(x)=0} \sum_{P\,:\,yx-\text{chemin}} CC(x) \times w(P) \Big) + \sum_{x/d^+_{GNG}(x)=0} BC(x) \tag{6}$$

où:

- $\sum_{x/d^+_{GNG}(x)=0} BC(x)$ est le coût total de fabrication des blocs (BC) x satisfaisant $d^+_{GNG}(x) = 0$, ce qui représente les blocs installés dans la conception.
- $\sum_{P\,:\,yx-\text{path}} CC(x) \times w(P)$ est le coût de calcul (CC) imposé par le bloc installé x, responsable de la réalisation du standard y.
- $\sum_{x/d^+_{GNG}(x)=0} \sum_{P\,:\,yx-\text{path}} CC(x) \times w(P)$ représente le coût total de calcul imposé par tous les éléments de traitement x installés dans la conception pour effectuer la fonctionnalité de la norme y.
- $\sum_{y/d^-_{GNG}(y)=0} \Big(\sum_{x/d^+_{GNG}(x)=0} \sum_{P\,:\,yx-\text{path}} CC(x) \times w(P) \Big)$ représente le coût total de calcul pour réaliser tous les standards considérés.

0.3.2 Une borne supérieure du nombre d'options de mise en œuvre

On va associer un vecteur X_v à chaque sommet v dans l'hypergraphe orienté H d'un système multi-standards qui contient L niveaux. Chaque entrée de X_v représentera le coût total de calcul résultant d'un choix particulier de mise en œuvre choisi pour implémenter le bloc v, où le coût est calculé via la fonction de coût de l'équation 3. Ce vecteur contient toutes les implémentations possibles de v. La dimension de X_v est donc exactement égale au nombre total d' options capables de réaliser v.

Dans le reste de cette partie, on note: $|X_v| = dim(X_v)$.
Les paramètres dont on a besoin pour former les entrées du vecteur X_v seront le coût de fabrication (BC), le coût de calcul (CC) et le nombre d'appels (NoCs).

Le vecteur X_v va être défini de façon récursive à partir des blocs de plus bas niveaux jusqu'à ceux de niveaux les plus élevés. Pour les blocs v dans le niveau 1, on a $|X_v| = 1$, parce que ces blocs peuvent être uniquement mis en œuvre en les installant eux-mêmes. Cette seule entrée dans X_v sera le CC du bloc v, ce qui représente le CC total imposé quand le bloc est installé lui-même. Après avoir défini X_v pour tous les blocs v tels que $l(v) \leq i$, le vecteur X_v où $l(v) = i+1$ est défini comme suit:

- Si l'on rencontre un hyperarc de type «OU» $e \in FS(v)$ et en supposant que $H(e) = \{r\}$ (donc $e = (T(e), H(e)) = (\{v\}, \{r\})$), alors chaque entrée dans X_r multipliée par $w(v, r)$ sera une entrée en X_v décrivant le coût total de calcul (CC) de l'une des options capable d'implementer v via r.

- Si l'on rencontre un hyperarc de type «ET» $e \in FS(v)$ et en supposant que $H(e) = \{s_{i1}, s_{i2}, \cdots, s_{in}\}$ (donc $e = (T(e), H(e)) = (\{v\}, \{s_{i1}, s_{i2}, \cdots, s_{in}\})$), alors : on choisit une entrée de chacun de $X_{s_{ik}}$, $k \in \{1, 2, \cdots, n\}$, on la multiplie par le nombre de fois que v appelle s_{ik} (ce qui est $w(v, s_{ik})$), puis on ajoute toutes les réponses obtenues pour tous $k \in \{1, 2, \cdots, n\}$. Cela formera une entrée de X_v. Par conséquent, il est évident que cet hyperarc impose $|X_{s_{i1}}| \times |X_{s_{i2}}| \times \cdots \times |X_{s_{in}}|$ options capables de réaliser v.

- Une option de plus est à considérer pour l'installation du bloc v lui-même.

On note Z_v le multi-ensemble obtenu à partir de X_v en listant simplement ses composantes.

Soit H un hypergraphe orienté d'un système multi-standards et $v \in V(H)$. $\forall\ e \in FS(v)$, on pose $U_e = \prod_{r \in H(e)} Z_r$.
Les composants de X_v seront alors identifiées par :

- $\forall e \in FS(v)$, $\forall a = (a_r)_{r \in H(e)} \in U_e$, $\sum_{r \in H(e)} w(v, r).a_r$ est une entrée en X_v.
- Une entrée supplémentaire dans X_v est le CC de v, qui représente l'installation directe du bloc v lui-même.

La dimension de X_v, $|X_v|$: D'après ce qui précède, on peut définir $|X_v|$ par:

$$|X_v| = \sum_{e \in FS(v)} \prod_{r \in H(e)} |Z_r| \ + \ 1, \tag{7}$$

défini de façon récurrente du plus bas au plus haut niveau.

On note u_i une borne supérieure de $|X_v|$ pour tout bloc v occupant le niveau i, c-à-d: $\forall\ v\ /\ l(v) = i, \quad |X_v| \leq u_i$. Une expression de u_i sera notre borne supérieure souhaitée. u_i sera ainsi définie récursivement du plus bas au plus haut niveau.
Les deux paramètres suivants seront utilisés:

$$M = \max_{v \in V(H)} (|FS(v)| + 1),$$
$$r = \max_{e \in E(H)} (|H(e)|).$$

On peut déduire une relation de récurrence comme suit:

$$\begin{cases} u_1 = 1, \\ \\ u_{s+1} = M(u_s)^r \quad \forall\ 1 \leq s < L. \end{cases} \tag{8}$$

Supposons que l'on veuille trouver u_s une borne supérieure de $|X_v|$ où $l(v) = s$. On peut alors faire les remarques suivantes :

- Il y a $M - 1$ hyperarcs tels que v soit le sommet parent (par définition de M).
- Chacun de ces hyperarcs contient un maximum de r sommets d'extrémités.
- Le pire des cas est réalisé lorsque tous les r sommets d'extrémités d'un hyperarc $e \in FS(v)$ sont dans le niveau $s-1$, ce qui imposera une borne supérieure plus large.

Il est à noter que la relation de récurrence de l'équation 3.6 peut être facilement résolue avec un simple induction sur s. On obtient alors : $u_s = M^{r^{s-2}+r^{s-3}+\cdots+r+1}$ ou encore :

$$u_s = M^{\frac{1-r^{s-1}}{1-r}};\ \ s > 1,\ \ r \neq 1. \tag{9}$$

Dans la partie suivante, nous allons exposer notre problème d'optimisation dont le but est de trouver l'une des options de mise en œuvre d'un terminal multi-standards ayant un coût minimum. A priori ce problème est un problème complexe en raison de la borne supérieure exponentielle atteinte pour le nombre total d'alternatives vu ci-dessus.

0.3.3 La complexité de notre problème d'optimisation

Notre objectif est d'identifier les opérateurs communs (COs) les plus appropriés permettant de mettre en œuvre un terminal multi-standards donné, et dont le coût (donné par la fonction de coût expliquée dans 0.3.1.4) est minimum. Ainsi, notre problème d'optimisation peut être décrit selon :

Une description générale des paramètres: Les paramètres d'une variance de notre problème doivent représenter une structure graphique H d'un système multi-standards. Ce qui suit est une liste des variables de notre problème d'optimisation.

1. une liste de tous les n sommets dans $V(H)$.
2. une liste de tous les m hyperarcs dans $E(H)$.
3. le nombre de niveaux L.
4. le nombre de blocs dans chaque niveau. Soient $a_1, a_2, \cdots, a_L$ le nombre de blocs dans les niveaux $1, 2, \cdots, L$.
5. Une liste des blocs occupant le niveau le plus haut (niveau L) qu'il est nécessaire d'implémenter.
6. Les valeurs numériques "CC & BC" de chaque bloc et le nombre d'appels (NoCs) sur chaque arc.

Question : Trouvez l'ensemble d'opérateurs $U \subseteq V(H)$ capables de mettre en œuvre le terminal multi-standards et dont le coût d'implémentation (donné par la fonction de coût) est minimum.

Rappelons qu'un problème d'optimisation peut être transformé en un problème de décision et qu'il n'est pas plus difficile que le problème d'optimisation correspondant. Le problème de décision correspondant à notre problème d'optimisation est formulé comme suit:

Une description générale des paramètres :
- les points de 1 à 6 dans la description des paramètres du problème d'optimisation (voir ci-dessus).
- une constante $B \geq 0$.

Le problème de décision se formule alors de la façon suivante :

Question : Peut-on trouver un ensemble d'opérateurs $U \subseteq V(H)$ capables d'implémenter les standards mis en jeu et dont le coût d'implémentation (donné par la fonction de coût) soit inférieur ou égal à B?

Nous avons alors établi le théorème suivant :

Théorème 3. *Le problème de décision précédemment décrit est un problème NP à condition que le nombre de niveaux L du graphe représentant l'équipement multi-standards soit majoré par une constante i, c-à-d $L \leq i$ pour $i \geq 0$.*

Preuve. Considérons une variance I de notre problème. L'objectif est de rechercher une solution S pour cette variance, puis d'essayer de déduire une relation polynomiale du nombre d'opérations nécessaires pour vérifier si la réponse pour I et S est «oui». Ceci implique l'étude du pire des cas qui correspond au nombre d'opérations maximum. Cependant, comme il est généralement impossible de déterminer exactement le pire des cas exact, on va examiner des scénarios qui sont pire que le pire des cas.
Considérons une structure graphique H d'un système multi-standards avec tous les paramètres nécessaires (la variance I) contenant L niveaux. Recherchons une solution S (une solution pour notre problème est un ensemble $U \subseteq V(H)$) et supposons que $|U| = k$ (c-à-d que l'on choisit k blocs par hasard). On devra alors vérifier les trois points suivants:

- Tout d'abord, il faut vérifier si la solution S peut mettre en œuvre l'équipement multi-standards: on montre qu'un maximum de $m(L-1)$ opérations est requis.
- Ensuite, il faut calculer le coût de l'option choisie, qui est caractérisée par les blocs sélectionnés dans U;
  Dans cette étape, on doit trouver le nombre de multiplications et d'additions requis dans le calcul de la fonction de coût. Un graphe généré (GNG) ressemblant à celui de la figure 3.7, noté W_{GNG}, correspondant au pire des cas de mise en œuvre:
  1. les k blocs choisis dans U occupent le plus bas niveau (niveau 1), ce qui génère les chemins les plus longs du niveau le plus élevé au niveau le plus bas.
  2. une connexion de type "ET" entre chaque bloc v et tous les blocs qui occupent un niveau inférieur de celui de v correspond au pire des cas, car elle impose un nombre de chemins maximum des niveaux les plus hauts aux blocs choisis dans U.

  Dans notre cas, nous sommes sûrs que la réalisation pratique ne sera pas plus complexe que le cas de la figure 3.7 considérée. C'est pourquoi on peut dire que c'est un cas encore plus complexe que le pire des cas.

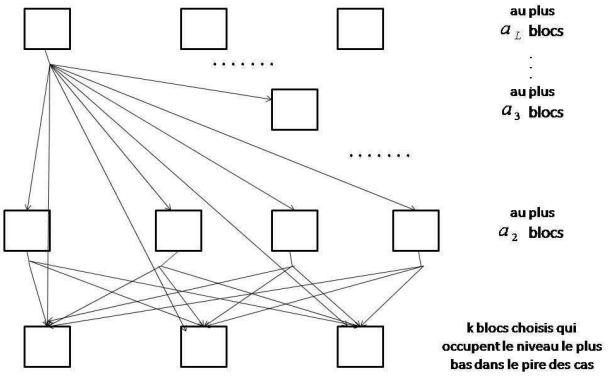

Figure 6: Graphe W_{GNG} généré représentant le pire des choix de mise en œuvre

Soit v un sommet de W_{GNG} tel que $l(v) = i$. On peut toujours trouver un chemin allant de tout sommet de $\Im$ à v traversant n'importe quelle combinaison et n'importe quel nombre de sommets de niveaux $L-1$ jusqu'à $i+l$ (probablement aucun sommet du tout), occupant tous des niveaux différents. Il faut noter que les chemins traversent les sommets dans l'ordre décroissant des niveaux du graphe que chaque sommet occupe. Soit v un sommet de niveau i dans W_{GNG}. On note n_i une borne supérieure du nombre de chemins de $\Im$ à v. On peut obtenir la relation de récurrence suivante:

$$\begin{cases} n_{L-1} = a_L \\ \\ n_{L-i} = n_{L-(i-1)}(a_{L-(i-1)}) + n_{L-(i-1)} \end{cases} \tag{10}$$

Soit $s = \max\{a_2, \cdots, a_L\}$. Par induction, on peut facilement conclure que n_{L-i} est de l'ordre de $O(s^i)$. Par conséquent $n_1 = n_{L-(L-1)}$, qui représente une borne supérieure du nombre total de chemins de $\Im$ à un sommet de niveau 1 dans W_{GNG}, est de l'ordre de $O(s^{L-1})$. Il faut noter que le nombre total de chemins de $\Im$ à tous les k blocs installés dans niveau 1 est au plus kn_1.

Le calcul des poids de chacun de ces kn_1 chemins est nécessaire (la longueur d'un tel chemin est au plus $L-1$) et ceci en cohérence avec la fonction de coût de l'équation 3.4. Ensuite son poids est multiplié par le coût de calcul (CC) du bloc correspondant installé. Ainsi, nous pouvons conclure que chaque chemin est associé au plus à L multiplications.
Puisque l'on a besoin d'une addition entre le poids d'un chemin et un autre, on aura $kn_1 - 1$ nombre d'additions dans le pire des cas.
Ainsi, au total, le nombre de multiplications et d'additions nécessaires pour calculer le coût correspondant sera inférieur à: $kLn_1 + (kn_1 - 1)$.

- Ensuite, il faut comparer le coût trouvé dans la deuxième étape avec B: une seule opération est alors nécessaire.

Le nombre total d'opérations requises pour vérifier la réponse «oui» de la variance I et la solution S sera donc :

$$m(L-1) + [kLn_1 + kn_1 - 1] + 1. \tag{11}$$

Si on désigne par b le $max\{s, k, m\}$, alors le nombre d'opérations nécessaires associées à l'équation 3.9 est une fonction en $O(b^L)$ (rappelons que $n_1 \in O(s^{L-1})$ et $L \leq i$). En outre, puisque $L \leq i$, on conclut que l'équation 3.9 est une fonction en $O(b^i)$, nécessitant ainsi un temps polynomial.

□

Puisque l'on sait maintenant que notre problème d'optimisation n'est pas un problème facile, il est nécessaire d'étudier des façons de le simplifier. Une possibilité réside dans l'exclusion d'un certain nombre de configurations permettant de réaliser l'équipement multi-standards. Ce point fera l'objet du chapitre suivant. Ensuite, nous proposerons un nouvel algorithme pour résoudre ce problème d'optimisation, en utilisant différentes notions de modélisation liées aux hypergraphes orientés, qui ne prennent en compte que les alternatives de mise en œuvre non-ignorées. Une analyse de complexité de cet algorithme par le pire des cas sera ensuite présentée.

0.4 Une technique d'optimisation des équipements multi-standards utilisant la notion d'hypergraphes orientés

Des travaux ont été menés par S. Gul [60] afin de résoudre ce problème d'optimisation. Etant donné la complexité du problème (dont la preuve a été apportée ensuite dans la section 0.3.3), une technique sous optimale (de type heuristique) a été à l'époque retenue. En effet, toutes les méthodes optimales connues nécessitent un effort de calcul dont la complexité augmente de façon exponentielle avec la taille du graphe. Cependant, dans ce travail, nous avons proposé un nouvel algorithme qui fournit la solution optimale. Sa complexité sera ensuite étudiée.

0.4.1 Exclusion de certaines configurations pour la recherche du coût minimal

Il a été mis en évidence dans la section 3.1.2 qu'il existe certaines alternatives capables de mettre en œuvre la conception pour lesquelles un bloc donné est installé avec quelques autres qui peuvent le construire.

Dans cette partie on va prouver que de telles configurations, dont les graphes générés ne correspondent pas à des G-chemins de la structure graphique d'un système mutli-standards, ne peuvent avoir le coût minimal. En effet, après avoir effectué les développements nécessaires, nous montrons que d'autres configurations moins coûteuse existent. Cela sera illustré sur un exemple et ensuite une généralisation sera fournie.

0.4.1.1 Un exemple

Un exemple est donné sur la figure 4.1. La configuration (1) représente un GNG d'une conception où l'implémentation de D et F est faite par les blocs de hauts niveaux $H\&I$ (par rapport à $J, K, L, \&I$), alors que dans configuration (2), les blocs de niveaux plus bas $J, K, L, \&I$ seront installés pour réaliser à la fois D et F. Quant au GNG de la troisième configuration, le choix est d'implémenter F en utilisant $J, K, L, \&I$ mais aussi d'installer les blocs de niveaux plus élevés $H\&I$ (qui peuvent être mis en œuvre par les blocs de niveaux inférieurs $J, K, L, \&I$) pour réaliser les fonctionnalités de D.

Remarquons que, dans la troisième configuration de la figure 4.1, il y a une partie dédoublée (DP) (qui comprend les fonctions $A\&H$) qui est utilisée par un chemin qui a besoin de ses fonctionnalités (le chemin traversant bloc D). Le deuxième chemin atteignant DP (traversant bloc F) considère plus de décompositions de certaines des fonctions à l'intérieur de DP (la décomposition de l'élément de traitement H).

On note d'autre part qu'il y a une partie non-dédoublée (UP) (la fonction I dans la figure 4.1) qui partage dans le calcul de CC(A), où A est le seul bloc de plus haut niveau dans DP.

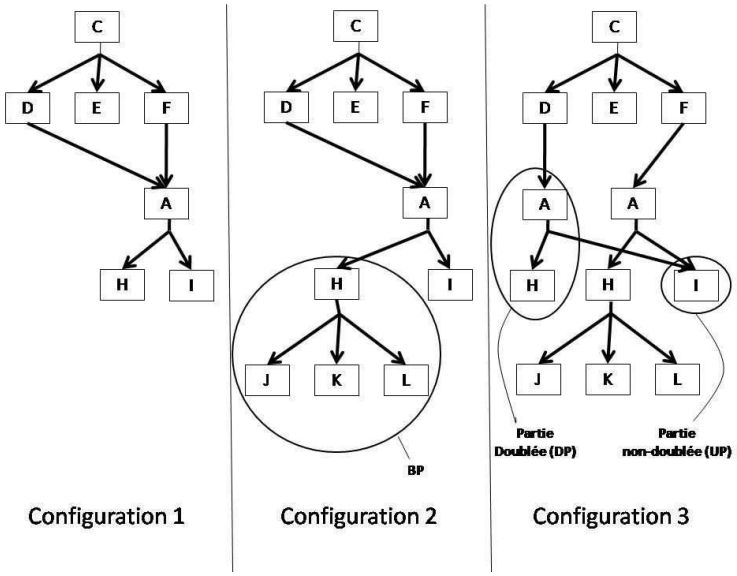

Figure 7: Trois configurations possibles

La question se porte maintenant vers les coûts de chacune de ces configurations. Le but est de démontrer que la configuration (3) ne peut en aucun cas avoir un coût minimum. Cela se démontre de la façon suivante. Tout d'abord, nous avons supposé que le coût de la configuration (2) était inférieur ou égal au coût de la configuration (1). En conséquence, nous avons alors démontré que le coût de la configuration (2) était obligatoirement inférieur au coût de la configuration (3). Ainsi, la configuration (2) est celle dont le coût est minimal. L'autre possibilité était de supposer que la configuration (2) avait un coût plus élevé que les configurations (1) et (3), et de prouver alors que, dans ce cas, la configuration (1) était la moins coûteuse.

0.4.1.2 Généralisation du principe l'exclusion

Dans cette partie, nous avons généralisé notre étude sur les possibilités de concevoir un design de coût minimum. En plus des parties dédoublées et non-dédoublées désignées par DP et UP, les notations suivantes ont été utilisées dans la généralisation: DUP désigne la combinaison de DP et UP; DUBP est une combinaison des fonctions de DUP avec la décomposition (désignées par BP) de quelques opérateurs dans DP. Par exemple dans le cas de la figure 4.1, nous pouvons remarquer que DUP est constitué des blocs A, H et I alors que A, H, I, J, K, et L forment ceux dans DUPB. Notons que H et I sont les blocs de DUP de plus bas niveaux; J, K, L et I sont ceux de plus bas niveaux en DUPB.

Ainsi pour généraliser, on va se référer aux trois configurations de la figure 4.2. D'une manière analogue à l'exemple précédent, la configuration (1) consiste en deux chemins realisés par les blocs de DUP de plus bas niveaux, tandis que la deuxième configuration comprend deux chemins mis en œuvre par les blocs de plus bas niveaux

de DUPB. Quant à la troisième configuration, l'un des deux chemins est mis en œuvre en utilisant les fonctions installées de DUP, et l'autre est réalisé par les opérateurs installés de DUPB. Nous avons alors démontré le théorème suivant:

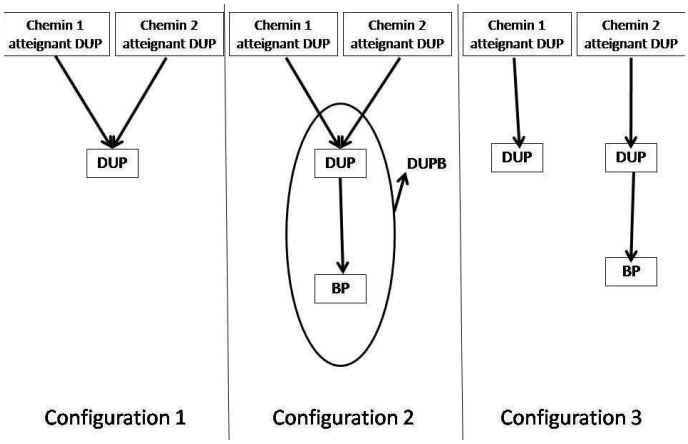

Figure 8: Les trois configurations dans la cas général

Théorème 4. *Une option de mise en œuvre d'un système multi-standards, dont le graphe généré comprend une partie dédoublée, ne peut jamais avoir un coût minimum.*

Pour le prouver, nous avons suivi la même démarche que celle utilisée dans l'exemple précédent.
Nous avons tout d'abord supposé que DP contient un seul bloc de plus haut niveau. La deuxième étape consiste à considérer que DP contient deux blocs de plus haut niveau. Dans ce cas, on a divisé DP en deux parties dédoublées dans lesquelles chacune contient exactement un seul bloc de plus haut niveau. Ensuite on a travaillé sur chaque DP séparément. On a remarqué que par induction, on peut étendre la preuve aux parties dédoublées contenant n'importe quel nombre de blocs dans le plus haut niveau.

Il faut noter que les deux premières configurations des figures 4.1 et 4.2 sont des G-chemins, ce qui n'est pas le cas pour la troisième. En conclusion, un concepteur recherchant une configuration de coût minimal peut ignorer toutes celles dont les graphes générés contiennent une partie dédoublée. Ainsi, il peut restreindre son étude aux configurations de mise en œuvre dont les graphes générés sont des G-chemins de racine $\Im$. Cette idée sera exploitée dans la partie suivante, où l'on propose un nouvel algorithme (en utilisant la théorie des graphes) qui peut résoudre le problème d'optimisation posé précédemment en ne traitant que les options de mise en œuvre dont les graphes générés sont des G-chemin de racine $\Im$, au lieu de traiter toutes les alternatives possibles.

0.4.2 Un algorithme de recherche de configuration à coût minimal

Notre algorithme de conception à coût minimal est noté MCD (Minimum Cost Design). Soit H un hypergraphe orienté représentant la décomposition d'un système multi-standards. On va définir l'entrée de notre algorithme par H_r l'hypergraphe orienté obtenu à partir de H en ajoutant un sommet r imaginaire à $V(H)$ et l'hyperarc E_r à $E(H)$, où $\{r\}$ est l'ensemble d'origine de E_r et $\Im$ représente son ensemble d'extrémités. Le sommet r joue le rôle d'un standard imaginaire qui occupe seul le niveau le plus haut. Les paramètres assignés aux entités de H_r seront:

- In english: CC, BC and NoCs for the entities of the directed hypergraph H_r (on the blocks and the arcs) remain the same for all the similar entities of the directed hypergraph H.
- CC, BC et NoCs pour les entités de l'hypergraphe orienté H_r (sur les blocs et les arcs) restent les mêmes pour toutes les entités similaires de l'hypergraphe orienté H].
- $w_{E_r}(r, v) = 1 \ \forall \ v \in \Im$

Dans notre algorithme lorsque l'on évalue le coût imposé par un G-chemin, on a besoin de rechercher tous les chemins de H allant de tous les standards de $\Im$ (blocs de plus hauts niveaux) à tous les blocs installés dans la configuration sélectionnée. Cela correspond à la recherche de tous les chemins du sommet r dans H_r aux blocs installés de la configuration donnée.

L'algorithme fonctionne de la façon suivante. Il sélectionne une option (illustrée comme un G-chemin), calcule son coût en utilisant la fonction de coût de l'équation 3.4, puis il le compare avec les autres coûts calculés précédemment, afin de le mettre à jour au cas où il serait inférieur à tous les coûts calculés jusque là. Ensuite, l'algorithme génère plusieurs configurations qui émergent à partir de celle finalement sélectionnée. La même procédure est suivie pour chaque configuration sélectionnée.

Le résultat de l'algorithme MCD correspond au G-chemin de coût minimal ainsi que son coût correspondant. A partir de ce G-chemin, on peut extraire les opérateurs communs qui doivent être installés dans la conception en sélectionnant simplement les blocs qui ont une étoile devant vide, donc atteignant notre objectif de trouver les opérateurs communs les plus adaptés qui permettront d'optimiser le coût.

Au cours des itérations de l'algorithme on va introduire, pour chaque G-chemin X sélectionné, un vecteur k_v associé à chaque sommet $v \in V(X)$ défini récursivement des sommets de plus hauts niveaux dans X vers ceux de plus bas niveaux où:

$$\begin{cases} \dim k_v = 1; \ \ v = r; \\ \dim k_v = \displaystyle\sum_{e \in BS_X(v)} \ \sum_{w \in T(e)} \dim k_w; \ \ v \neq r; \end{cases} \tag{12}$$

Chaque composante de k_v représentera le poids d'un chemin de r à v, et la dimension de k_v dim k_v correspondra au nombre de ces chemins.

Plusieurs variables ont été introduites dans l'algorithme. On va expliquer l'intérêt de quelques unes d'entre elles.

- Q est un ensemble dans lequel les sommets de l'option sélectionée X seront appelés progressivement, en traversant les boucles de l'algorithme. A chaque étape, l'algorithme sélectionne un élément u dans Q qui satisfait $l(u) = \max\{l(w); w \in Q\}$, en raison de la façon selon laquelle le vecteur k_v est défini.
- M est un ensemble dans lequel les G-chemins générés de H_r de racine $\{r\}$ seront appelés.
- Une variable R_P est introduite pour contenir le coût total du $p-ieme$ G-chemin sélectioné.
- S est une variable dans laquelle on accumule le coût d'un certain G-chemin de H_r de racine $\{r\}$.
- A est un ensemble qui contiendra tous les sommets v de degré extérieur zéro dans le G-chemin X sélectionné.
- SMin est une variable entière qui contiendra le plus faible coût d'un G-chemin.
- K est une variable qui contiendra le G-chemin de H_r de racine $\{r\}$ ayant le coût le plus faible parmi ceux testés.

Voici les étapes complètes de l'algorithme "MCD":
Procedure$(H_r, CC(v), BC(v), NoC(v))$
begin
$M = \{(\{r\}, \phi)\}$, $R_p := 0$, $p := 1$, $D := \phi$;
repeat
select and remove $X \in M$
if $X = (\{r\}, \phi)$
go to step U | **1 :Skip the cost evaluation of this imaginary option** X
end-if
$S := 0$, $A := \phi$;
$k_r := k_{1r} := 1$ **dim** $k_r := 1$;
for each $v \in V(X)\backslash\{r\}$ **do**
$k_v = 0$ **vector,** **dim** $k_v := 0$; | **2 :Initialization of** k_v **vectors for** X
end-for
$Q = \{r\}$;
repeat
select and remove $v \in Q$
for each $E \in FS_X(v)$ **do**
begin
for each $h \in H(E)$ **do** | **3**
begin
$Q := Q \cup \{h\}$

$i := 1$
repeat
if $k_{ih} \neq 0$
$i := i + 1$
end-if
until $k_{ih} = 0$
4: Indicate the index i of the first 0 component of k_h

if $v \neq r$
for each $E \in BS_X(v)$
for each $w \in T(E)$
dim k_v := **dim** k_v + **dim** k_w
end-for
end-for
end-if
5 : Compute dim k_v using eq. 12

$j := 0$
repeat
$k_{(i+j)h} = k_{(j+1)v} \times w_E(v,h)$
$j := j + 1$
until $j :=$ **dim** k_v
6 :Find weight of each rh-path via E

if $d_X^+(h) = 0$
$l := i$
repeat
$S := S + CC(h) \times k_{lh}$
$l := l + 1$
until $l = i +$ **dim** k_v
$A := A \cup \{h\}$
end-if
7 :Multiply found weights in step 6 by CC(h)

end-for
end-for
until $Q = \phi$
repeat
select and remove $v \in A$
$S := S + BC(v)$
until $A = \phi$
8:Add BC of all installed blocks to total CC

$R_p := S$
if $p = 1$
SMin $:= R_p$
$K := X$
end-if
9:R_1=the first cost found, indicated to be so far the least

$p := p + 1$
$R_p := 0$
10:Initialize R_p variable to 0, which will include next cost

if $p > 2$
if $R_{p-1} <$ **SMin**
SMin $:= R_{p-1}$
$K := X$
end-if
end-if
11:Update SMin & K to associated least cost option

STEP U
for each $u \in V(X)$ / $d_X^+(u) = 0$ **do**
begin
for each $E \in FS_{H_r}(u)$ **do**
$M := M \cup \{X + E\}$
end-for
end-for
until $M = \phi$
end-procedure.

12:Generate G-paths of H_r **from** X

Dans la partie suivante, nous allons procéder à une analyse du pire des cas de cet algorithme pour estimer sa complexité. Cette analyse de la complexité de calcul donnera une borne supérieure pour les ressources requises par l'algorithme.

0.4.3 Complexité de calcul de l'algorithme MCD

Soit H_r l'hypergraphe orienté (une donnée de l'algorithme contenant L niveaux, sans tenir compte du niveau r, avec $|V(H_r)| = n$ et $|E(H_r)| = m$. Soit a_k le nombre de sommets qui occupent le niveau k dans H_r. Considérons de plus les paramètres suivants:

- $E_i = \bigcup_{v \in l(L-i+1)} FS(v)$; Evidemment $E_L = \phi$.
- $t = \max_{i=1,\cdots,L-1} |E_i|$.
- $W = \{v / d_{H_r}^+(v) = 0\}$.
- $s = \max_{i=2,\cdots,L} a_i$.
- $d = \sum_{v \in V(H_r)} d_{H_r}^-(v) = \sum_{e \in E(H_r)} |H(e)|$

Dans la thèse, nous avons démontré les points suivants:

1. Le nombre maximum d'hyperarcs que n'importe quel G-chemin de H_r de racine $\{r\}$ peut comprendre n'exède pas $n - |W|$.
2. Le nombre total de G-chemins de H_r de racine $\{r\}$, contenant au plus k hyperarcs chacun, est une fonction de l'ordre de $O(t^{k-1})$. Par conséquent, le nombre total des d'options générées par l'algorithme sera de l'ordre $O(t^{n-|W|-1})$, puisque chaque G-chemin de H_r de racine $\{r\}$ contient au plus $n - |W|$ hyperarcs.
3. Une borne supérieure de la dimension de n'importe quel vecteur k_v est de l'ordre de $O(s^{L-1})$. Ceci avait déjà été prouvé en utilisant un raisonnement similaire à celui similaire à celui du deuxième point de la preuve de la section 0.3.3.

Il est clair que le coût de l'initialisation à l'étape 2 de l'algorithme est de l'ordre $O(n)$. On suppose que chaque opération de sélection et d'élimination des ensembles M, Q, ou A ainsi que d'insertion dans M, Q, ou A a un coût unitaire.

Chaque sommet est inséré et retiré de l'ensemble Q au plus une fois, parce que la sélection d'un élément u de Q satisfait $l(u) = \max\{l(w); w \in Q\}$. Donc, on peut conclure que dans la troisième étape de l'algorithme, chaque hyperarc de X ne sera examiné qu'une seule fois (c-à-d la première fois que l'hyperarc est sélectionné) et à chaque fois qu'il est sélectionné, tous ses sommets d'extrémités seront examinés. Ainsi, les étapes 4 à 7 de l'algorithme seront exécutées $\sum_{e \in E(H_r)} |H(e)| = d$ fois. Les étapes 4, 6 et 7 ont des complexités de l'ordre $O(s^{L-1})$ chaque fois que l'on entre dans la boucle "répéter $\cdots$ jusqu'à $Q = \phi$", car dans toutes ces étapes on a besoin d'explorer les composantes d'un certain vecteur k_v. Au contraire, l'étape 5 sera exécutée $\sum_{v \in V(H_r)} d^-_{H_r}(v) = d$ fois tout au long des itérations de la boucle "répéter $\cdots$ jusqu'à $Q = \phi$", puisque la dimension de chaque sommet dans X sera calculée seulement une fois.

L'étape 8 a une complexité de l'ordre $O(n)$, car un maximum de n sommets existent dans A. Les étapes 9, 10 et 11 sont d' une complexité négligeable (juste quelques affectations et comparaisons). L'étape 12 a une complexité de l'ordre $O(m)$, car l'un de m hyperarcs au maximum peut être chaque fois ajouté à l'option selectionée. Toutes les étapes de 1 à 12 ont une complexité de l'ordre $O(t^{n-|W|-1})$.

Finalement, on peut conclure que la complexité de l'algorithme MCD est de l'ordre $O((ds^{L-1} + d + n + m)t^{n-|W|-1})$ ce qui est équivaut une complexité de l'ordre $O(a^{n-|W|+L-1})$. Ainsi, une borne supérieure exponentielle est atteinte pour les ressources requises par l'algorithme MCD. Notons que ceci est une analyse du pire des cas et ne représente pas nécessairement le nombre exact atteint en pratique.

L'algorithme MCD a été implémenté en langage C. Nous l'avons appliqué sur plusieurs exemples afin d'établir ses performances. Nous avons commencé par un exemple d'un hypergraphe orienté et puis nous l'avons complexifié en ajoutant au fur et à mesure un hyperarc. Nous avons remarqué que le nombre d'options de mise en œuvre (et par conséquent le temps d'exécution) augmente rapidement avec l'augmentation du nombre d'hyperarcs, et cette augmentation diffère en fonction de l'endroit où l'hyperarc est ajouté. Enfin, il était évident que notre approche est plus complexe que la technique stochastique précédemment sélectionnée par S.Gul (qui ne nécessite pas beaucoup de temps d'exécution puisque c'est une technique heuristique). Mais cependant l'algorithme MCD est évidemment capable de fournir une solution exactement optimale au contraire des techniques heuristiques qui donnent une solution proche de l'optimum.

0.5 Conclusion

Ce travail se situe dans le domaine de la radio logicielle et plus précisement dans l'identification de structures communes pouvant aboutir à la réalisation de terminaux reconfigurables et flexibles. Cette approche appelée paramétrisation peut être abordée de façon théorique ou pragmatique. Seule la déclinaison théorique a été étudiée dans ce manuscrit. Il s'agit d'un travail de recherche sur l'optimisation d'équipements

SDR multi-standards en utilisant la théorie des graphes.

Nous avons commencé le manuscrit en présentant le problème de conception lié au domaine de la radio logicielle restreinte dans le chapitre 1. L'approche de type *Velcro* encore majoritairement utilisée ne sera bientôt plus viable. D'où la nécessité d'introduire de nouvelles méthodes d'optimisation de conception, en particulier d'équipements multi-standards, qui vont se généraliser. Partant de ce constat, nous introduisons l'approche de conception par paramétrisation et en particulier son approche théorique . Cette approche consiste à étudier tous les appels de fonctions du système selon plusieurs niveaux de granularité en utilisant une représentation du problème sous forme d'un diagramme [50]. Cette approche graphique a été ensuite associée à un problème d'optimisation en proposant une fonction de coût [49] qui doit être minimisée. Cette fonction de coût évalue le coût de chaque option de mise en œuvre sélectionnée parmi toutes les configuration possibles d' un équipement SDR multi-standards.

Des techniques stochastiques donnant des solutions proches de l'optimum ont été proposées dans [60] pour résoudre ce problème d'optimisation, car il a été intuitivement considéré comme un problème complexe.

Dans le chapitre 2 les deux théories, essentielles pour la suite de notre travail, à savoir celle des graphes et de la compléxité, ont été présentées en détail.

La structure graphique d'un système SDR multi-standards introduite dans le premier chapitre a été dans un premier temps modélisée comme un hypergraphe orienté dans le chapitre 3. Nous avons ensuite fourni une suggestion graphique d'une option de mise en œuvre comme un hypergraphe orienté obtenu à partir de la structure graphique d'origine, appelé un graphe généré. En outre, nous avons proposé une forme alternative de la fonction de coût présentée dans le premier chapitre en utilisant différentes définitions des hypergraphes orientés. Ensuite, nous avons menée une étude dans le but d'estimer le nombre de configurations possibles décrites par l'hypergraphe orienté associé au système multi-standards. Nous avons montré que ce nombre atteint une borne supérieure exponentielle, caractéristique d'un problème très complexe. Enfin, nous avons présenté une preuve détaillée démontrant que notre problème d'optimisation est NP mais à condition que le nombre de niveaux dans la structure graphique ne dépasse pas une certaine constante, ce qui est généralement considéré le cas.

Dans le chapitre 4 nous avons prouvé que lorsque l'on cherche une configuration de coût minimum, il est possible d' ignorer celles dans lesquelles on trouve une duplication dans leurs graphes générés. Nous pouvons restreindre notre recherche aux configurations ou modèles dont les graphes générés sont des G-chemins. Cela nous a permis de proposer un nouvel algorithme à coût minimal (MCD) pour résoudre notre problème d'optimisation, en utilisant des notions de modélisation liées aux hypergraphes orientés. L'algorithme fournit le G-chemin correspondant à une configuration à coût minimum, à partir duquel nous pouvons extraire les opérateurs communs

destinés à être installés dans l'équipement multi-standards mutualisé. Nous avons analysé la complexité de calcul de l'algorithme MCD ; sa borne supérieure est exponentielle, signifiant que le problème reste complexe.

Comme il s'agit d'une analyse du pire des cas, il semble évident que dans la plupart des cas pratiques, une complexité moindre sera nécessaire. C'est pour cela nous avons appliqué notre algorithme MCD à divers exemples pour illustrer l'évolution de sa complexité, après avoir développé un code pour MCD en langage C. Les résultats montrent que MCD nécessite un effort de calcul important qui augmente rapidement au fur et à mesure que l'on augmente le nombre d'hyperarcs dans l'hypergraphe orienté. Bien que notre algorithme nécessite une plus grande complexité que les techniques heuristiques stochastiques, MCD est capable de fournir une solution optimale. Enfin, il est important de noter que le temps d'exécution de l'algorithme MCD n'est pas un obstacle car il est exécuté une seule fois avant de concevoir l'équipement multi-standards .

Les perspectives de ce travail de thèse sont les suivantes :

- La solution proposée vise à partager des opérateurs communs entre plusieurs stnandards. Par conséquent, le problème d'ordonnancement et d'allocation des ces opérateurs est primordial. Pour cela, une démarche méthodologique doit être proposée pour organiser les exigences de l'utilisation de chaque opérateur commun. Il pourrait être nécessaire de dupliquer certains opérateurs dans la conception.
- La complexité du problème d'optimisation presenté dans ce manuscrit peut être enrichie pour savoir si le problème est encore plus complexe, c'est à dire en prouvant qu'il s'agit d'un problème NP-complet (si possible).
- L'exclusion de certaines options de mise en œuvre a certainement réduit la complexité de notre problème d'optimisation. Une question malgré tout se pose : dans quelle mesure la complexité de notre problème a été réduite en supprimant ces options.
- L'idée d'ignorer certaines options de mise en œuvre, dont les graphes générés ne correspondent pas à des G-chemins, peut être exploitée sur toute autre solution trouvée dans l'avenir pour notre problème, que ce soit déterministe ou bien un outil d'optimisation heuristique, aboutissant à une technique moins complexe en examinant un moins nombre d'alternatives capables de mettre en œuvre le système multi-standards.

Part II

Ph.D. Dissertation

General Introduction

1 Background and context

The recent years have witnessed an enormous proliferation of standards especially in wireless communications and broadcast television, just to mention few. These standards form just the basis for a sophisticated electronic device which will keep on evolving with technology innovation, each with the potential to sell in very high volume. Thus, radio system designers nowadays focus their attention on developing multi-standard systems that can successfully communicate with different systems using different air-interfaces.

Current habit is to support several communication standards through dedicated self-contained complex components, known as the *Velcro approach* [1, 41], which is however difficult to be updated with changes in the standards. The need for an adaptable and flexible terminal which is capable of being upgraded with the latest innovations, as well as to operate with all supported standards in different geographical regions of the world, is ever increasing.

In the beginning of the 90's, Mitola introduced the concept of *SoftWare Radio* (SWR) [6] as an architecture which uses general-purpose hardware that can be programmed or reconfigured in software [2], to support many different transmission standards on a common platform. It emerged from demonstrations in military research to become a cornerstone of the third generation for ubiquitous global communications. SWR technology allows to roam over various networks in different geographical environments supporting multiple heterogeneous applications. It was considered to afford more flexibility in accepting multiple waveforms and services at a reasonable cost and complexity. Such reconfigurable architectures are able to support new standards and to add new functionalities as soon as they are discovered because they possess sufficient flexibility, instead of the frequent redesign imposed by non-flexible equipments which is definitely costly and time consuming.

One key idea of the SWR concept is to exploit the common signal processing operations between the different standards which are intended to be implemented on a common platform. In order to attain higher flexibility and adaptability, it was necessary to first identify the most appropriate commonalities between the various applications to be supported, and then to find the optimal way to reuse some hard-

ware and software modules without affecting the system's performances. In this sense, a technique called *parametrization* is introduced [35, 36]. It relies on the common operator's approach which allows several standards to use the same components and share their cost, where the behavior of the common aspects can be modified by a simple parameter adjustment imposed by a software module [37]. Parametrization represents a methodology to design SWR and is considered to be a crucial optimization process to design flexible multi-standard systems.

Two approaches are considered in the context of parametrization, namely the *Theoretical approach* and the *Pragmatic approach*, which are complementary approaches to identify the most appropriate common operators inside and between the different standards. The former approach, which forms the keystone of this work, helps find the global optimality using a suggested graph structure of multi-standard systems in [49, 50] to construct an optimal design. As for the pragmatic approach, it identifies and forms common operators from the architecture point of view, which can be invoked in the graph structure of the theoretical approach. It creates common operators by merging like-looking architectures, as was done to create the Dual Mode FFT operator in [46] and the common operators based on the Linear Feedback Shift Register (LFSR) in [47] and [48]. The next section describes the scope and the main objectives of this thesis.

2 Scope and objectives

A graphical approach for designing flexible multi-standard radio systems is proposed in [50] which describes in a diagram the interrelationships between the different components in the system, by decomposing the communication blocks into different granularity levels. This approach lies within the theoretical technique of parametrization. It's a mathematical model in which it's required to perform the necessary granularity adjustments, by solving the optimization problem associated with it. In fact, a cost function is formulated in [49] using two selected parameters; the Building cost which is the cost of a Processing Element (PE) paid only once, and the Computational cost which is the time taken by a PE to perform a certain function and is paid every time it's called. This cost function is required to be optimized to its minimum value possible by using appropriate optimization techniques. Stochastic search optimization approaches which give near-optimal solutions were selected in [60] to solve this problem instead of using deterministic search techniques, because the related optimization problem was intuitively considered to be a complex one. Since the problem is formulated in a graphical diagram in [50], our aim is first to reformulate and model it theoretically using graph theory. In light of this modeling comes our second objective aiming to find or create a new optimization tool to solve this optimization problem.

Graph theory [62] is rapidly moving into the mainstream of mathematics mainly because of its applications in diverse fields which include telecommunication (Viterbi DEcoding, radio channel assignment, $\cdots$), biochemistry & biology topics, electrical

engineering (communication networks), as well as computer science (algorithms and computation). It is the study of graphs used to model pairwise relations between elements from a certain collection. There might be no distinction between the related elements in a graph. However, elements of a graph may be directed from one element to another in which case it is called a digraph. Hypergraphs [68] and directed hypergraphs [69] are generalizations of graphs and digraphs respectively.

The task of providing a theoretical characterization using graph theory for the various notions of the multi-standard system helps to explore the system's different design possibilities that the corresponding graphical diagram exhibits. Our final objective in this thesis is to choose the most convenient design that optimizes the cost imposed by the proposed cost function in [49]. In this case it is necessary to study the complexity of our optimization problem, which was not studied or examined except in this work, to guide us in the search for the most adequate graph theoretical tools capable of solving it. Once an appropriate optimization technique is proposed, which is certainly preferred to yield an exact-optimal solution instead of a near-optimal one, an optimal multi-standard design can be constructed by choosing the most adequate common operators in the most convenient granularity levels from the associated graphical diagram.

3 Thesis outline

This thesis is composed of four chapters. In chapter 1, we introduce the SWR technology in general and outline some of the challenges associated with it, especially those imposed by the digital processing converters, which entail the emergence of its feasible practical version called the Software-defined Radio (SDR) presented next in this chapter. In addition, we highlight an important area for designing SDR equipments which is the parametrization technique that fits in the common operator's approach for identifying processing commonalities between the different standards to be supported in the design. We will present the two categories of the parametrization technique where we briefly present the *pragmatic approach* with some examples, and then introduce the *theoretical approach* in some details. In the theoretical approach of parametrization, we explain first how SDR multi-standard systems are explored at different levels of granularity in a diagram which provides all the design alternatives. Then, we present a cost function equation suggested in [49, 50] which evaluates the cost of one specific design. Accordingly, we state an optimization problem of minimizing the cost, where our final objective will be to solve this problem. To do this, we will need to study its complexity after having theoretically modeled the problem using graph theory, which steps will be accomplished in the following chapters.

Chapter 2 mainly presents two necessary theories for our work; graph theory and the theory of complexity. It consists of three parts. In the first part, we define the different aspects of graphs (whether directed -called digraphs- or not) with their particular and generalized versions (hypergraphs and directed hypergraphs), as well

as present some of the definitions and theorems associated with each. Afterwards, we introduce the theory of complexity which helps in evaluating the computing effort associated with problems and algorithms. In this context, we explain different classes of problems classified as P, NP, or NP-complete problems according to their solvability conditions. The final part looks for some graph theory problems and applications to different real-world situations and highlight the complexity imposed by each. This part will provide several examples on graph-theoretical modeling to different network connection and telecommunication topics.

Chapter 3 is responsible of providing a theoretical model for the various aspects related to the problem of designing an SDR multi-standard system, we mention modeling the graph structure of an SDR multi-standard system and providing a formal expression of the previously introduced cost function using graph theory and more precisely, using directed hypergraphs. As our final aim is to choose one of the combinations of common operators from different granularity levels which optimizes the cost, so the second point in this chapter is to explore the number of possibilities to design an SDR multi-standard system. In fact, we find an upper bound for the total number of alternatives capable of implementing the standards to be supported in the design in order to have an idea of the complexity of the problem we're dealing with. This complexity issue is studied and examined afterwards in this chapter. Even though it seemed that our optimization problem is complex, we didn't have a precise idea of its complexity in the general case. However, we were able to prove that this problem is a Nondeterministic Polynomial-time (NP) problem but under a certain identified condition on the number of decomposition levels in the suggested graph structure of an SDR multi-standard system.

In chapter 4, we first present a quick view on two previously highlighted optimization techniques in [60] to solve our optimization problem; the exhaustive search which provides an exact-optimal solution and the simulated annealing which selects a near-optimal one, where the latter approach was considered to be a better technique for the problem in hand because it requires much less computing effort. After having proved in chapter 3 that our optimization problem is NP under a certain specified constraint, we perform in this chapter an exploration on the various options of implementation and prove that some of them can be ignored or skipped when trying to find a solution for this problem. This helped to propose a new algorithm as a next step in this chapter, using different modeling notions related to directed hypergraphs, which examines all the options of implementation for designing an SDR multi-standard system except the ones lately excluded. This algorithm gained importance being an approach which provides an exact-optimal solution, unlike the heuristic techniques. Afterwards, a computational complexity analysis of this algorithm is performed yielding an upper bound on its time execution requirements, where it seemed that our proposed algorithm needs a non-negligible computing effort. After having developed a code for our algorithm in C language, we used this algorithme to solve the optimization problem for several generic examples with the costs being arbitrarily chosen, in order to explore its performance skills. The difference between our algorithm and the previously suggested ones for this problem is

finally highlighted.

Finally, we end this thesis with a summary of the achieved results and an outline of the possible future work in this field.

Chapter 1

Parametrization technique for Software-Defined Radio

Contents

This chapter presents the SoftWare Radio (SWR) technology developed for designing flexible multi-standard terminals, where a terminal in this thesis is considered as a mobile station or a basestation. However, SWR raises an important number of challenges from the design point of view which entails the emergence of the Software-Defined Radio (SDR) system, the feasible version of SWR.
Parametrization is a proposed design methodology for SDR terminals, which relies on the concept of exploring common aspects inside and between the different supported standards. Parametrization can be tackled from two different approaches: the theoretical and pragmatic approach. Both approaches have the same objective of identifying the appropriate set of cohabiting commonalities, but only contradict in the way of dealing with the problem. All these ideas will be elaborated in the present chapter. The theoretical approach of parametrization, however, constitutes the keystone of our work.

1.1 SoftWare Radio

The tremendous development of the wireless communication is driven by the accelerating rate of emergence of new standards and protocols. In order to comply to the accelerating rate of technology innovation and the predicted technological change, wireless system manufacturers nowadays should focus on providing systems that can adapt to the changes as they occur by upgrading them to the latest innovations as soon as they are discovered. This is a more favorable solution than changing the whole design with every new development, since frequent redesign is so expensive and time consuming.

A challenge is to create future-proof radios whose hardware and software combinations are capable of being updated with the proliferation of new standards, techniques, and technologies. This is achieved by exploiting the commonalities among different standard modules, which entails the replacement of the analog modules by digital ones. A main reason for replacing analog with digital signal processing is the possibility to reconfigure the system "softly", thereby enabling the implementation of different air interfaces on a given platform. This yields an open-architecture based radio system whose most of the dedicated functions are executed by software.

The current implementation of wireless communication equipments completely in hardware faces several problems. One of these problems is that there is a significant difference between each generation of network (from second generation 2G to 3G then further onto 4G) and thus legacy handsets may be incompatible with newer generation network. Indeed, the difference in the air interface and protocols across various geographical regions form another problem, as it inhibits the deployment of global roaming facilities causing great inconvenience to users who travel frequently from one continent to another.

The SWR concept was first introduced in the literature around the 1990's by Joe Mitola [6] to refer to the class of radios which extend the evolution of programmable hardware, increasing flexibility via increased programmability. This is accomplished by a complex interaction between a number of common issues in the radio design. In this context, the same piece of hardware would be able to perform different functionalities at different times.
SWR technology has generated tremendous interest in the wireless industry, military implementations as well as in commercial and civil applications [4, 38, 39].

SWR was introduced as a solution to the dedicated hardware problem enabling the implementation of radio functions in networking equipment and user terminal as software modules running on a generic hardware platform. Such implementation helps reduce the cost and the size of systems which are required to support a wide range of existing and future wireless technologies [5]. The SWR system has the following features:

Reconfigurability: SWR architectures are reconfigurable terminals [3, 4] that are able to adapt to changes. They allow implementation of different standards on the same terminal, where multiple software modules exist and the required standard is run by just downloading the suitable software module.

Global connectivity: The software implementation of the different air interface standards helps in realizing global roaming facility. In this case, the terminal can be upgraded, whenever it's incompatible with the available network technology in a particular region, by a simple download for an appropriate software module [1].

The possibilities to design software radio architectures range from "Velcro" approach to the "Very Fine Grain" approach. The so known "Velcro" approach is to realize a juxtaposition of several standards and the reconfiguration is simply realized by a switch from one to another, headed by a set of parameters [41]. In such approaches there's no share of common resources between the standards which opposes the flexibility concept, and the complexity increases with the increase in number of standards. On the contrary, a design following the "Very Fine Grain" approach is based on manipulating small size operators (which will be invoked several times) to support different standards. This design is flexible but its drawbacks are that it's highly sequential and time consuming. However, an optimal way of realizing a multi-standard terminal is to identify the appropriate common functions and operators inside and between the standards, referred to the "parametrization" approach. The parametrization technique will be tackled in details in this chapter.

SWR has several beneficial and interesting characteristics since it can be:

1. a multi-band system which supports more than one frequency band used by a wireless standard (e.g., GSM 900, GSM 1800, $\cdots$),
2. a multi-standard system that supports more than one standard.
3. a multi-service system which provides different services (e.g., voice, data, video streaming, television broadcast, $\cdots$).
4. a multi-channel system that simultaneously supports two or more independent transmission and reception channels.

SWR imposes several software realization requirements on the different functions of a transceiver communication chain. This entails some modifications on the nature of the different stages of a transceiver architecture, which face various challenges and obstacles. These obstacles will be highlighted in the next section after having introduced the different stages of a conventional transceiver architecture.

1.2 Conventional transceiver architecture

Fig. 1.1 represents a conventional transceiver terminal. It consists of four stages namely Data processing, Baseband processing, digital processing conversion, and Radio Frequency (RF) front-end. The following includes several words on some of these stages.

- Data processing is responsible of manipulating the data packets, i.e analyzing the input data and synthesizing the ones to output.
- The baseband system, which precedes the RF front-end, is responsible for end-user data encoding/decoding.
- Digital processing conversion constitutes both forms of conversion tools; The Analog to Digital Converter (ADC) which encodes analog signals into digital ones, and the Digital to Analog Converter (DAC) which performs the reverse process.
- The main functions of the RF front-end include down and up conversion, channel selection, interference rejection and amplification. More precisely, the functionalities of each of the receiver and transmitter side of the *RF front-end* are summarized by the following:

  Transmitter side of the RF-front end takes the signal from the DAC, converts it to the transmission radio frequency, amplifies the signal to a desired level, limits the bandwidth of the signal by filtering in order to avoid interference and feeds the signal to the antenna [37].

  Receiver side of the RF-front end converts the signal from the antenna to a lower center frequency (using the RF Band Pass Filter (BPF)) such that the new frequency range is compatible with the ADC, filters out noise and undesired channels (by the Low Noise Amplifier (LNA)), and amplifies the signal to the level suitable for the ADC (using the Automatic Gain Control (AGC)).

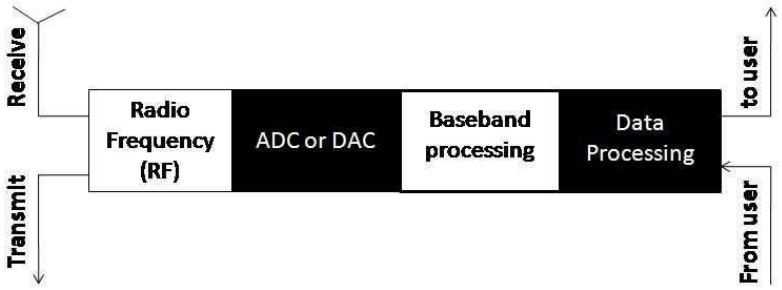

Figure 1.1: Conventional Transceiver

A transceiver is referred to as a software radio if its communication functions are realized as programs running on a suitable processor. This ideal SWR directly samples the antenna output. However, many challenges raise in such designs especially on the level of the RF front-end and the digital processing convertors. The RF front end should be directly suitable for different frequencies and bandwidths required by the different standards that the SWR equipment is intended to support. A main objective during the design of an optimal RF-front end is to attain a trade-off between power consumption and dynamic range. Challenges imposed by the digital processing convertors will be addressed in the next section. Thus, a practical version of SWR, called Software-Defined Radio (SDR) was presented in [3]. SWR is defined

for comparison purposes only. In an SDR system, the received signals are sampled after a suitable band selection filtering, i.e employing Intermediate Frequency (IF) sampling, thus incorporating some Analog Front End (AFE). This feasible SWR version will be presented in details in the following section.

1.3 The Feasible SoftWare Radio design

The radio communication system has witnessed several major evolutions during the last century. In this section, we highlight the emergence of the SDR systems and afterwards stress on the challenges imposed by the digital processing convertors, which form some of the major obstacles for the creation of the ideal SWR design. Finally, we will describe and analyze the SDR architecture.

1.3.1 Emergence of Software-Defined Radio

Nowadays, radio system designers aim to develop flexible multi-standard terminals that support as much air interface standards as possible in order to cope with the daily accelerating rate of technology innovation. For this purpose, many standards are involved and realizing the several analog components (associated to each of the supported standards) becomes a very challenging task. As a simple example, in the receiver side of a multi-standard system, the down conversion from the RF to the baseband requires several Local Oscillators LOs, each associated to a certain carrier frequency compatible with the relative frequency of the selected standard. This imposes the presence of several dedicated analog parts, thus occupying a huge complexity.

As a solution, the researchers have proposed to digitize the analog signal in RF instead of digitizing after the conventional conversion done from RF to baseband. In other words, in the receiver side, the signal has to be directly digitized by an ADC and then fed to low complexity, high speed and reconfigurable channelizers which extract the digital signal as dictated by one's needs. Thus, in such architectures, the convertors are set right beside the antenna [29] with all the filtering and most of the signal gain being done in a digital signal processor, thus increasing the digital processing to enhance the reconfiguration attained by software. This is what is called the ideal SWR illustrated in Fig. 1.2a. However, the need for this software radio architecture raises a number of technical challenges. Most important of these challenges are related to ADC and DAC and most particularly, the receiver ADC forms the most challenging component limiting the choice of the RF front-end architecture [1]. The digitizing of the received signal has to be achieved with a high performance and wide-band ADC. The major problem that the SWR technology faces is that the actual ADCs are not able to cope with that very high frequency signals. So, a more realistic SWR architecture is illustrated in Fig. 1.2b, where the digital part of the transceiver is placed as close as possible to the antenna but some RF front-end still remains crucial, at least for amplifying and filtering. In this case, the ADC and DAC are placed between the stages of channel modulation, at an IF and the digitization is

done at IF instead of RF (IF < RF). This is the feasible SWR case referred to as SDR.

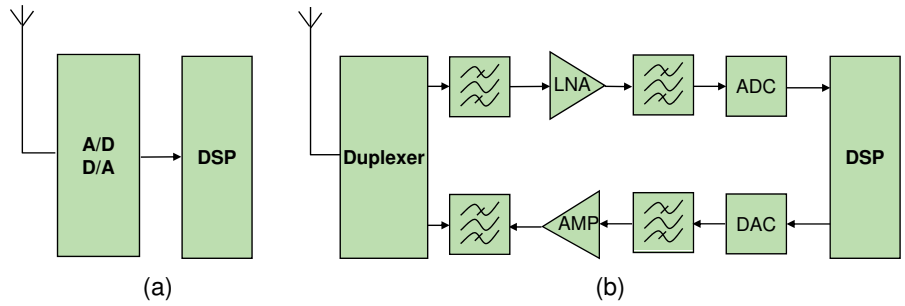

Figure 1.2: a.) Ideal SWR transceiver architecture b.) SDR realistic transceiver architecture

Software-defined and hardware reconfigurable radio systems have attracted more and more attention recently because they are expected to come to the market in the context of dramatic changes in the worlds' information technology and usage environments. Important work has been done on software-defined radio in [3, 4, 7, 27], and on the reconfigurability issue of SDR in [9, 10, 11, 12, 13]. Expertice are required for each particular aspect of the signal processing chain of SDR. Thus, separate work is achieved on each of the sample rate adaptation in [14, 15, 16], the RF-front end design in [17, 18, 19, 20, 21], and the channel coding in [22, 23, 24]. In an SDR design, the receiver section is more complex than the transmitter. The ADC conquers an important area in SDR receivers, where the work concentrates on digitizing the analog signal in the receiver as close to the antenna as possible. The state-of-the-art in ADC technology can be found in [25, 26].

1.3.2 Challenges imposed by the ADC and DAC

The ADC and DAC are among the key components for the SWR design [30]. They identify the bandwidth, the dynamic range [(1)] and have a deep impact on the power consumption of the radio. However, the ADC is one of the most critical part in a software radio design. The number of bits in the ADC specifies the upper limit for achievable dynamic range. The physical upper bound for capabilities of ADCs particularly can be derived from Heisenberg's uncertainty principle [6]. The state of the art ADCs for wireless radios sample at rates of about 100 million samples per second and quantize the signal with 14-bit resolution. These performances don't fulfill the desired level of the required wide dynamic range mainly when the ADCs have to cope with signals with large bandwidth. Besides, according to Nyquist-Shannon sampling theorem, the sampling rate must be at least two times the highest frequency component of the analog signal to avoid the loss of information [31], which

[(1)] The ratio of a specified maximum level of voltage to the minimum detectable value.

imposes stringent requirements on the highest frequency to digitize by the ADCs.

In conventional radio architectures, the conversion from analog to digital signals is done at the baseband. However, the closer are the digital processing blocks (i.e the ADC and DAC) to the antenna, the more flexible is the architecture to accepting the desired multiple frequency and multiple channel bands. An ideal SWR consists of digital processing blocks placed as close as possible to the antenna, but any practical implementation still needs some analog parts of RF front end, and the design of a reconfigurable RF part remains a very complicated issue [1, 4].
So, to be transformed into an ideal SWR architecture, it must employ the digital processing blocks right beside the antenna which is currently feasible for very low centered frequency band, but rather impossible at a low cost and a low power consumption for higher RF frequencies. Thus for the feasible SDR architecture, the typical place for the ADC and DAC is between the stages of channel modulation, at an intermediate frequency.

1.3.3 The Software-defined Radio architecture

The SDR design is split into two parts, one digital part referred to as Digital Front End (DFE) and one analog called an Analog Front End (AFE). The receiver of an SDR design, is illustrated in Fig. 1.3. The AFE selects a continuous signal and shifts all its bandwidth from the high frequency (RF) to some lower frequency (IF) which is suitable for the available ADC. As for the DFE, this is a part of the receiver which realizes functionalities on a processor digitally (using a Digital Signal Processor (DSP)) that were initially realized by means of analog signal processing. This includes channelization, sample rate conversion, etc. Channelization performs all the necessary tasks to select the desired channel, including conversion to baseband, channel filtering, etc. Sample rate conversion is done digitally by generating the clock which changes the sample rate to adapt to the multiples of symbols dictated by the different standards. This is just to mention few.

To conclude, we can say that the stringent requirements imposed by SWR are relaxed by SDR. The today's attained system has to be at half digitized and the second half can't be digitized at low cost and low power consumption before having available advanced ADCs able to provide an extreme dynamic range and very high sample rate, in order to digitize the bandwidth of all the supported bands and directly after the antenna.

The interest to reduce the power consumption, the cost, and the size of the radio while reaching the expected performances and taking full advantage of technology improvements was a motivation to search for optimization aspects for flexible radio system designs. The goal of this thesis is to focus on the global SWR issues and to develop new theoretical techniques of optimization of the SDR multi-standard flexible systems using *Graph Theory*. This will provide the options to the SDR designer

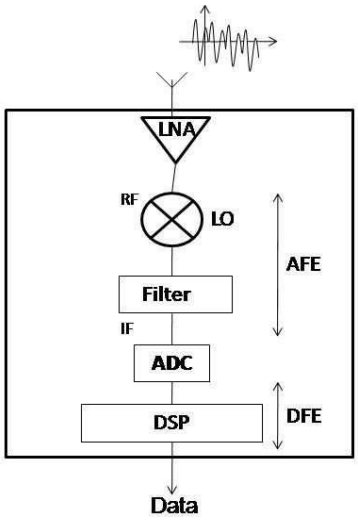

Figure 1.3: The Software-Defined radio receiver architecture

to orient his design either towards Velcro or gate level primitive elements.

In this work, the SWR's problematic will be tackled from the viewpoint of a research of the appropriate commonalities inside and between the different standards. In this context, an important technique called parametrization was introduced in [35] as a methodology to design SDR, which aims to optimize the resources use in the SDR system. This technique will be highlighted in the next section. The parametrization approach is divided into two approaches, namely the theoretical and the pragmatic approach, which can describe the conception and the realization of a parameterizable SWR system. The theoretical approach forms the foundation of this thesis.

1.4 Parametrization technique

The conventional approach to the implementation of multi-standard systems is to realize a juxtaposition of several transceiver chains each dedicated to an individual standard and the reconfiguration is simply realized by a switch from one to another. This conventional approach called *Velcro approach* [1] is not flexible as most of the hardware needs to be replaced whenever the characteristics of the interface change. Indeed, this approach doesn't exploit any common resources between the supported standards and thus the complexity increases as the number of standards increases. On the contrary, an approach which exploits the commonalities among various signal processing operations for different standards offers a promising solution to the design of an optimal multi-standard architecture which balances between flexibility and complexity. In this context, the parametrization technique is introduced in [35, 36], whose concept is to reuse some hardware and software modules without affecting

the system's performances. According to the methodology of parametrization, the common aspects of the different standards become one common processing element which could be installed in the device and executed by a simple call.

Parametrization represents a methodology to design SWR. It can be considered as a crucial optimization process to design flexible multi-standard systems. This approach aims at designing multi-standard systems made of certain operators (or functions) in which their behavior can be simply adjusted by a simple parameter modification [37].

A very promising procedure is to consider parametrization as a technique based on two approaches which are the theoretical and the pragmatic approach. In the theoretical approach of parametrization, the different modules of the several standards intended to support in the design are illustrated in a diagram that represents the hierarchical level of each module functionality. Afterwards, an optimization process has to be proposed in order to identify optimal levels of granularity, from which components can be considered as "common communication blocks" enabling their reuse by several applications. It's in this context where the theoretical approach tackles the discussion of the Common Operators (COs) approach.

In contrast, the pragmatic approach is a practical approach developed to identify or create possible COs that will be, nevertheless, invoked in the theoretical graph illustration. Thus, both approaches converge to the same objective i.e to identify the best set of commonalities among the different standards, leading to a flexible reconfigurable multi-standard conception. They are considered to be complementary approaches to one another. In this section, we will highlight the COs approach, then provide a quick view on the pragmatic approach and a long detailed discussion on the theoretical approach of parametrization. However, the foundation of our thesis is in the theoretical parametrization approach context, where we use graph theory in order to identify the most appropriate COs from the most convenient granularity levels of the graph illustration. Various necessary graph theory definitions and notations will be detailed in the next chapter.

1.4.1 The Common Operators technique

The *common operator* technique consists in identifying common elements based on structural aspects. In reference to the definition of the pragmatic approach presented in 1.4.2, it's intended to design COs independently of the calling functions and thus of the supported standards. These designed COs might be called by distinct functions several times all along the SDR terminal. The COs technique is therefore considered to be an *open technique* [36] for the creation and development of the common aspects, since they are defined without any previous knowledge on the set of transmission modes and standards to be realized by the conception. In other words, a CO can be defined by an operator which can be reused by each function or processing requiring its functionality, independently of the application context.

As an example, the Fast Fourier Transform (FFT) has been considered as a CO in [42, 43], since it was shown that many important tasks of a communication equipment, such as filtering, channelization, Orthogonal Frequency Division Multiplexing (OFDM) modulation, etc, can be implemented through FFT. Indeed, frequency domain implementations for different families of algorithms which have the same performances as the time domain ones could also be investigated. This FFT CO can be used by any function that necessitates the use of Fourier transform which is what the concept of common operator is about.

Now we'll refer to the common operators technique from the theoretical approach view. Let's consider the graphical breakdown of the transmission chain of the multi-standard terminal introduced in [41] and developed in the discussion provided in 1.4.3, where some processing blocks are achieved by lower level processing operations and the granularity level of the considered components is decreased step by step up until the level of primitive operators e.g. adders, multipliers, etc. The goal is to identify a level of granularity at which common operators could be selected to implement the processing elements of the communication standards. The idea is not to select the maximum number of common elements, because the latency of systems would exceed certain limitations in such cases [40]. In fact, the optimization in terms of hardware or software resources depends on the performance in terms of delay or execution time and thus what we seek is the best cost-performance trade-off. In this context, it's important to determine the ideal levels of granularity to design a multi-standard equipment, neither the Velcro approach which employs complex dedicated communication components (but is highly parallel), nor the basic logic cells implementation which is very sequential (but has lower complexity and thus higher flexibility).

As a consequence of the CO technique, common parts will be reused several times which imposes scheduling issues at run time. Some scheduling considerations have been partially addressed in [**?**] and other relevant work can be found in [44, 45]. However, we don't go into the details of scheduling in this thesis as it is a major research topic by itself and might be further addressed by other PhD students in the future.

1.4.2 The Pragmatic approach of parametrization

The pragmatic approach is a technique which identifies and creates COs. It's divided into two stages. The first stage, called the Existing Search, consists in identifying in literature and from previous work some operators potentially recognized to be common. The second stage is the Constructive Search, which consists in the handmade building of COs. In this stage, one tries to merge like-looking architectures step by step to form common operators of higher levels.

As explained in 1.4.1, FFT was considered as a promising CO. Under the pragmatic approach concept, a reconfigurable architecture of a common FFT operator able to operate over two different domains (Complex field ($\mathbb{C}$) and Galois Field (GF)) was designed in [46]. This dual mode FFT, called DMFFT operator, which is used in different contexts, is able to reconfigure its hardware for the several required mode of operation. In other words, this operator needs to be parameterized by appropriate parameters (for example the length of the transform, the base of the discrete transform (exponential, etc $\cdots$)) in order to perfectly meet the requirements of each application. This is exactly what the origin of the "parametrization" term is about.

Another strong candidate for the common operators is the Linear Feedback Shift Register (LFSR). Two developed LFSRs architectures namely the Reconfigurable LFSR (R-LFSR) and Extended Reconfigurable LFSR (ER-LFSR) were presented in [47] and [48] respectively. These two architectures were developed from an improved classical LFSR structure. They were able to replace several functions (filtering functions, etc $\cdots$) that can be derived from LFSR operations.

Another pragmatic approach design is suggested in [51] which merges two widely used algorithms in wireless communication systems: Viterbi and FFT algorithms. The performance of this common FFT/Viterbi operator considering various tradeoffs as complexity and power consumptions is also discussed in [52]. Notice that although FFT and Viterbi algorithms seemed to be completely different when comparing their performed functions but however, when explored in the parameterization context, functional and architectural similarities were pointed out in [51].

Note that the exploration and selection of the different COs like FFT and LFSR is based on a formal approach detailed in [49, 50]. It's this theoretical formal approach which we seek to improve in our thesis work using *Graph Theory*.

1.4.3 The Theoretical approach of parametrization

A theoretical approach for designing flexible multi-standard radio systems is proposed, which consists in exploring such designs at different levels of granularity and selects the convenient level depending on each designer's needs. This is translated into a graph structure of the multi-standard system to be introduced in 1.4.3.1, which describes the interrelationships between the different components in the system. This graph representation provides all the options of implementation capable of realizing the multi-standard system. However, a cost function which calculates the cost of any selected one of these options is suggested in [49], and will be presented in 1.4.3.2.

1.4.3.1 Graph Modeling of SDR systems

The first step in the theoretical approach consists in elaborating a model for an SDR system intended to support several standards as a diagram representing the step by step breakdown of the different modules at different levels. This SDR multi-standard system's graphical representation, called the *graphical approach*, is introduced in [49, 50]. Each node in the figure represents a block or an elementary Processing Element (PE). In order to perform the functionalities of this PE, it can be installed by itself in the design, as a unified non divisible block, or it can be realized by some lower-level building blocks. It's necessary in such approaches to distinguish between the different dependencies of the nodes of different levels. For this two node dependencies, the "OR" and the "AND" dependencies, are essential to clearly illustrate the implementation needs of each block and describe the type of connection between blocks of higher levels and others in lower levels.

A node of a higher level, called a parent node, may have dependencies with nodes of underlying levels, called descendant nodes. An "OR" dependency (left part of fig. 1.4, direct arrow) means that only one of the descendant nodes (B or C) called several specific times is necessary to implement the parent node (A). On the contrary, an "AND" dependency (right part of fig. 1.4, inverted Y connection), signifies that all the descendant nodes via the "AND" dependency (B & C) are needed to implement the parent node (A) accompanied with certain number of calls. The concept of number of calls will be further explained in 1.4.3.2. These higher to lower dependencies mean that we can perform the required task of block A using block A itself in level n (realized as a dedicated block whether an Application Specific Integrated Circuit (ASIC) or a program) instead of using blocks in level $n-1$ which needs less time but has much higher building cost. On the contrary, realizing the functionalities of block A using blocks of lower levels (block(s) B or/and C) will decrease the installation cost but increase the execution time of the system. It should be noted that until this point, no strict assignment of level exists and the only important information is the relative level between nodes that are interconnected, rather than their absolute level. However, we will provide a formal definition of the notion of an absolute level of a block in chapter 3.

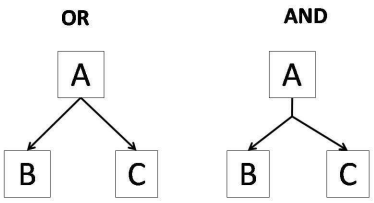

Figure 1.4: "OR" and "AND" dependency

In this way and using these two node dependencies, a generalized figure example can be drawn corresponding to a conceivable multi-standard system as shown in Fig. 1.5.

Such figures describe the various functionalities of the top level block standards to be supported in the design passing through several granularity levels (two standards denoted by S & T in the case of Fig. 1.5).
In Fig. 1.5, the "OR" dependency from $B4$ shows that it can be implemented through any one of $C2$ or $C3$. An "AND" dependency, as the one pointing from S to $A1, A2$, & $A3$, means that all the three descendent nodes are needed to implement the functionality of the parent node S. Note that in some cases, a parent node may have both "AND" and "OR" dependencies with its descendants, like the ones pointing from the block $A2$.

As a concrete example of the use of common operators, to realize an equipment that supports the standard S, it is possible to implement it as a unified non-divisible block. When all the standards intended to support are built in this manner, then this will correspond to the *Velcro solution*, where in order to change the standard it will be necessary to completely switch to another block for all the processing associated with the other standard. Another possibility to implement the standard S is to install nodes $A1, A2$, & $A3$ in the terminal. We can notice that now if the equipment wants to switch to standard T, $A3$ may be retained and thus save a lot in terms of reconfiguration compared to *Velcro case*. This is the primary interest of the theoretical approach of parametrization to be proposed in this thesis. One can even use lower level blocks to build the functionalities of S and T.

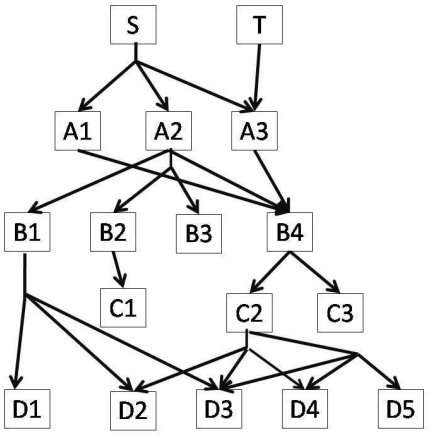

Figure 1.5: Generalized figure corresponding to a conceivable breakdown of two standards S & T

A more realistic simplified graphical structure of an SDR multi-standard system is shown in Fig. 1.6 supporting two different air interface standards. It represents a simplified version of the complete graph with only few important building blocks from transmitter side only, shown for illustration purposes. Of course, the complete graph would be huge showing all the nodes. Note that the building of the figure is a task in itself as researchers always try to identify new operators that could be

common to several communication "blocks" within a given standard, or across several standards (using pragmatic approach). These newly developed common blocks would be invoked in the figure and thus lead to its modification. It is not in the purpose of this thesis to develop complete graphs. In all our work, we will just present conceivable graph illustrations or some figures provided by other previous studies.

First selected standard in Fig. 1.6 is the Wireless-Fidelity (Wifi), used for wireless connection and now almost all laptops and other handheld devices come with Wifi built-in. Second selected standard is the Universal Mobile Telecommunications Systems (UMTS). It's a third-generation (3G) broadband, packet-based transmission of text, digitized voice and video.

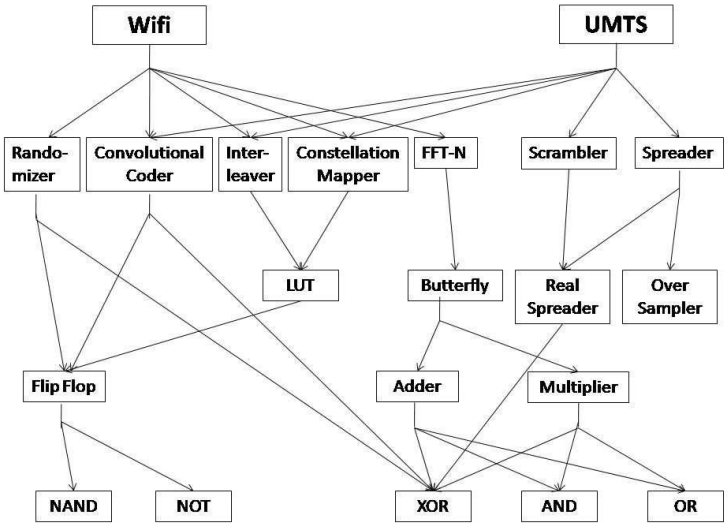

Figure 1.6: Global structure of a multi-standard graph (supporting Wifi and UMTS) - transmitter side

The roots of this diagram at the top level/coarsest grain level, represent the functions which realize the physical layer processing of the different standards to be supported by the radio (Wifi and UMTS in the case of Fig. 1.6). Each processing element (PE) occupies a certain layer depending upon its granularity level, where more complex PEs have higher granularity levels than less complex ones that can form their functionalities. As we move down to further lower levels in the figure, we pass through various granularity levels and finally reach at the primitive elements level (the lowest level). There might be many intermediate levels in between the top level and the very lowest. The tasks of some of the blocks in Fig. 1.6, including

randomizer[2], convolutional coder[3], Flip Flop[4], constellation mapper[5], interleaver[6], FFT-N[7], LUT[8], scrambler[9], & spreader[10] are highlighted.

This diagram offers a pictorial view of the numerous alternatives that are able to do the same task. One trivial alternative of implementation is to install in the design all the primitive operators occupying the lowest level (NAND, NOT, XOR, AND & OR), but such a design isn't flexible (requires huge amount of time as will be noticed later in this chapter) and thus we aim to increase the granularity level of the selected operators to intermediate levels. The goal of this approach is to provide the options to the designer, to select a set of operators each of which occupies a certain level of granularity, as dictated by his needs. Now the questions that come to mind are: Which one of these different alternatives of implementation of the multi-standard system is the best? Best according to which criteria? To answer these questions, first we need to assign weights i.e costs to the different entities of the diagram and then derive a cost function which gives a cost value for any possible option of implementation. Afterwards, certain optimization tools should be selected in order to choose the alternative which has the minimum cost.
A cost function is suggested in [49]. In the next subsection, we'll explain this proposed cost function and provide an example to illustrate its calculation process. As for the optimization process, the state-of-the-art selected optimization algorithms as well as our new optimization tools will be presented in following chapters.

1.4.3.2 An Objective Cost Function

In order to characterize a "quality" to any required solution, one needs to have some criteria to evaluate it. These criteria are expressed as some functions of the decision variables, which are called objective functions. In an analogous way, we will need first to identify the variables (or the parameters) of our problem, and then state our

[2] "A randomizer is a device used to invert the sense of pseudorandomly selected bits of a bit stream to avoid long sequences of bits of the same sense.

[3] "A convolutional coder is a type of error-correcting code in which (a) each m-bit information symbol (each m-bit string) to be encoded is transformed into an n-bit symbol, where m/n is the code rate ($n \geq m$) and (b) the transformation is a function of the last k information symbols, where k is the constraint length of the code.

[4] "Flip Flop is an electronic circuit that can assume either of two stable states

[5] "A constellation diagram is a graphical representation of the symbols used to encode digital data on a communication channel, particularly in phase- or quadrature-amplitude modulation systems

[6] "The Interleaver transmits multiple independent data streams over a single circuit, where one byte from each input stream is taken at one time and framed in a time-division sequence.

[7] "A Fast Fourier Transform (FFT-N) is an efficient algorithm to compute the Discrete Fourrier Transform (DFT) and its inverse. N stands for the number of inputs

[8] "Lookup Tables (LUT) is an array or matrix of data that contains items that are searched. Lookup tables may be arranged as key-value pairs, where the keys are the data items being searched (looked up) and the values are either the actual data or pointers to where the data are located.

[9] "Scrambler is an electronic device that scrambles telecommunication signals to make them unintelligible to anyone without a special receiver.

[10] "Spreader is a telecommunication system that transmits images of objects (stationary or moving) between distant points.

objective function(s).

A. Cost Parameters

There can be many cost parameters to assign to an SDR multi-standard system equipment. Among them, the two selected cost parameters in [49] to be associated with PEs of the graphical representation of an SDR multi-standard system are the *Building Cost* (BC) and the *Computational Cost* (CC), where:

- The BC stands for the cost of the building PE capable of computing a function. It is just paid once during the useful life of the radio system. This means that even if it is required to call a PE several times in the equipment, the BC associated with element will not be multiplied. Re-use of PEs thus provides an idea of cost reduction.
- The CC is considered to be the time taken by a PE to compute a function. This cost has to be calculated every time a PE is invoked or called by higher level PEs.

In this context, it is necessary to tag one more parameter to the arrows in the figure, called the *Number Of Calls* (NoCs), which is a multiplicative factor needed to represent the number of times a descendent module is called. In fact, the graph representation of any system consists of various granularity levels [53]. The task at higher granularity level can be performed with the help of components that are at lower granularity level but associated with certain number of calls. These necessary number of times a processing element B at lower granularity level will be called to perform the task of the component A at higher granularity will be a number attached to the arrow joining the two blocks, denoted by NoC(B).

Indeed, some communications between components may be also needed, including an additive *communication cost*. In this parameters' suggestion, the communication cost was not considered and was neglected.

Note that the BC can stand for the number of multiplications, number of additions, number of gates, number of execution cycles, area, etc $\cdots$. All depends on the concerns of the designer of the SDR system. As for the CC, usually it's referred to the execution time required by a PE, since this is a cost that has to be paid every time the block is incurred.

In general, a block having higher granularity level has a higher manufacturing or building cost than the ones with lower hierarchy that it calls. On the contrary, the execution time of a node with higher hierarchy is usually less than all the executions needed to be done to make the same calculation using the lower hierarchy nodes.
As a simple example, we consider the concrete case of a hardware implementation (a Field Programmable Gate Array (FPGA) or an ASIC implementation) concerning

the filter processing, which can be performed in many design options. Some designers might choose to parallelize their architectures in order to gain in speed of execution but at the cost of an additional area (building cost). This is the cost of the higher granularity level element. However, in order to save in terms of area, it is possible to call several times a single PE that maybe has lower hierarchy, with a much lower area cost and may also have a lower CC but which has to be called several times. The resulting CC might become more expensive accompanied with the several calls. Hence, it's evident that what we need is a compromise between fine and coarse granularity in the figure in order to attain the best complexity-speed of execution trade-off.

Fig. 1.7 is a very simple figure representing the breakdown of a block S up to two lower levels. PEs are tagged with BC/CC e.g. block A is tagged with 50/20 i.e BC of A is 50 while its CC is 20. Arrows are tagged with NoCs e.g the $\times 30$ entity on the arrow between block A and block B means that the functionality of A requires 30 times that of B. It's to be noted that these numerical values are arbitrarily chosen and follow a certain logic on relationships between BCs and CCs of higher to lower hierarchy blocks (as highlighted previously).
In Fig. 1.7, we see that block A can be used for a BC of 50 and a CC of 20, or an alternative design to realize A is to implement the three blocks $B, D, \& E$ for a total BC of $5 + 10 + 20 = 35$ and a CC of $1 + 2 + 10 = 13$ (if we consider a sequential execution only one time). But, since it's necessary to call each of $B, D, \& E$ $30, 20, \& 5$ times respectively to perform the functionality of A, then we get a total CC equal to $1 \times 30 + 2 \times 20 + 10 \times 5 = 120$.

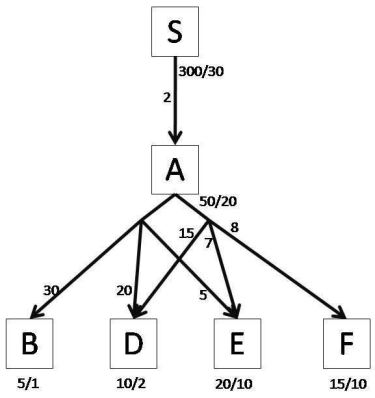

Figure 1.7: A simple figure showing the break-down of block S up to 2 lower levels

B. A Cost Function equation

In this section, we will describe the development of the suggested objective cost function. Based on the parameters already introduced (BC and CC), it's obvious that our aim is to minimize the total building cost of the system as well as the total computational cost. In this context, our cost function will be a bi-objective function.

These two objectives are conflicting in nature because reducing the BC will increase the total CC and vice versa. By associating the costs to the blocks and arrows, the problem turns into an optimization problem.

The first objective is to minimize the total building cost. This can be written as:

$$min \sum_i BC_i.N_i \tag{1.1}$$

where $\sum_i BC_i.N_i$ represents the total Building Cost of all the nodes implemented in the design, and $N_i \in \{0, 1\}$ depending on whether block B_i is installed in the design or not.

As for the second objective, i.e to minimize the total computational cost of a multi-standard SDR system containing N standards, it can be expressed as:

$$min \sum_n \sum_k CC_k((S_n)_{n\in\{1,2\cdots,N\}}) \tag{1.2}$$

where

- $\sum_k CC_k((S_n)_{n\in\{1,2,\cdots,N\}})$ stands for the total CC imposed by one of the N standards, S_n.
- $\sum_n \sum_k CC_k((S_n)_{n\in\{1,2,\cdots,N\}})$ is the total CC of all the N standards together.

This problem enters the family of multi-objective optimization problems. A multi-objective optimization typically arises in various engineering modeling problems, financial applications, and other problems where the decision maker chooses among several competing objectives to satisfy [54]. A multi-objective optimization problem can be written in the following form:

$$min\{f_1(x), f_2(x), \cdots f_k(x)\} \qquad s.t \quad x \in \Omega$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ are (possibly) conflicting objective functions and $\Omega \in \mathbb{R}^n$ is the feasible region. These objective functions may be commensurable (measured in the same units) or incommensurable (measured in different units). In general, the objective functions related to engineering optimization are incommensurable.
For consistency, all the maximization problems of the type max f_i are transformed into equivalent minimization problems $\min(-f_i)$. The goal of multi-objective optimization is to simultaneously minimize all of the objective functions.

Since it is often assumed that objective functions compete (or conflict) with each other, it is possible that there is no unique solution that optimizes all objectives at the same time. Indeed, in most cases there are infinitely many optimal solutions.

An optimal solution in the multi-objective optimization context is the one where there exists no other feasible solution that improves the value of at least one objective function without deteriorating any other objective. This is the notion of Pareto optimality [54, 55, 56, 57].

The easiest and perhaps most widely used method to handle a multi-objective optimization problem, is the weighted sum approach. Initial work on the weighted sum method can be found in [58] by Zadeh (1963) with many subsequent applications. Heuristic methods are also used for multi-objective optimization; Suppapitnarm et al. applied simulated annealing to multi-objective optimization [59], and multi-objective optimization by Genetic Algorithms can be found in Goldberg (1989), Fonseca and Fleming (1995), and Tamaki et al. (1996) among others.

The weighted sum approach was considered in [60] to define the objective cost function because of its popularity and simplicity. It consists in aggregating the different optimization functions in a single function. This method takes each objective function and multiplies it by a fraction of one, the "weighting coefficient" which is represented by w_i. The modified functions are then added together to obtain a single cost function, which can be easily solved using any method which can be applied for single objective optimization. Mathematically, the new function is written as:

$$min \sum_{i=1}^{k} w_i f_i(x) \qquad s.t \qquad x \in \Omega$$

where $w_i \geq 0, \forall i = 1, 2, \cdots, k, \quad \& \quad \sum_{i=1}^{k} w_i = 1.$

The weighted sum approach does nevertheless possess some disadvantages. It is an approach particularly sensitive to the setting of the weights. Thus, one needs to determine the appropriate weights, which requires enough information about the problem when usually this is not the case. The optimal solution will depend on the relative values of the weights specified. For example, if one is trying to maximize the strength of a machine component and minimize the production cost, and if a higher weight is specified for the cost objective compared to the strength, the solution will be one that favors lower cost over higher strength. Despite its disadvantages, the weighted sum approach proved to be an efficient and promising method in many applications of multi-objective optimization problems.

Returning back to our problem and as we previously mentioned using the weighted sum method, our bi-objective cost function which combines the two incommensurable objective functions of equations 1.1 and 1.2 will have the form:

$$Cost = (\overline{w} \sum_{i} BC_i.N_i + \sum_{n} \sum_{k} w_n CC_k((S_n)_{n \in \{1,2^{\cdots},N\}})) \tag{1.3}$$

where $\overline{w}$ stands for the weight given to the total BC of the system while w_n is the weight associated to the CC of executing standard n. w_n may refer to the cost of a

communication standard in function of its rate of activation compared to the other standards supported by the SDR system, as it's assumed that the standards supported by the design are not executing simultaneously.
Note that it's up to the designer to select the weights that suit his interests. For example, for some designers BC is more significant than other designers who are more concerned with the CC.

Finally, and after specifying the number of standards to be supported by the multi-standard SDR system, the solution for an optimal design consists in minimizing:

$$\mathbf{C_{SDR}} = \min_{bool(S_n)_n} (\overline{w} \sum_i BC_i.N_i + \sum_n \sum_k w_n CC_k((S_n)_{n\in\{1,2\cdots,N\}})) \tag{1.4}$$

where the constraint $bool(S_n)_{n\in\{1,2\cdots,N\}}$ checks that all the standards can be implemented in the corresponding SDR design.

Solving our problem will make us achieve our goals of finding a solution that balances between economy and computing efficiency. Although this cost function seems to be simple and straightforward, the problem still remains very difficult and complicated in practice. In all our following work, the weights $\overline{w}$ and w_n in the cost function of equation 1.3 will be neutralized to 0.5 unless otherwise stated, because they can vary according to the concerns of each designer.

In the rest of this part, we will explain the computation process of the cost function in equation 1.3. Let's reconsider the simple graph structure of Fig. 1.7 and compute the implementation cost of block S at different levels of implementation. As a first choice, suppose that S is going to be installed by itself in the design as a unique nondivisible block (Velcro approach). Then the cost to be paid to perform the functionality of S in this case will be:

$$\begin{aligned} Cost_{(Using\ S)} &= BC(S) + CC(S) \\ &= 300 + 30 = 330. \end{aligned} \tag{1.5}$$

Now, suppose that a designer chooses to use a block of lower level than S (the A block) to implement the block's S functionality. Block A will be installed in the design and will be invoked $NoC(A)$ times (2 times) in order to perform the required tasks. Consequently, the cost becomes:

$$\begin{aligned} Cost_{(Using\ A)} &= CC(A) \times NoC(A) + BC(A) \\ &= (20 \times 2) + 50 = 90. \end{aligned} \tag{1.6}$$

The task of S can be performed with the aid of components that are at even lower granularity level, lower than that of A. Here we have two options: either use B, D, & E or select the operators D, E, & F.

First we'll consider the former case. The total CC to perform the functionality of one A block using this option will be:

$$\begin{aligned} CC(A)_{(Using\ B,D,E)} &= CC(B) \times NoC(B) + CC(D) \times NoC(D) + CC(E) \times NoC(E) \\ &= 1 \times 30 + 2 \times 20 + 10 \times 5 = 120. \end{aligned}$$

Now since we need to call the tasks similar to those of A twice, then the total CC that has to be paid to realize S from B, D, & E operators will be:

$$\begin{aligned} CC(S)_{(Using\ B,D,E)} &= CC(A)_{(Using\ B,D,E)} \times NoC(A) \\ &= 120 \times 2 = 240. \end{aligned}$$

The final total cost of this choice will be to add the total CC of S (using B, D, & E) with the BC of each of B, D, & E only once. So:

$$\begin{aligned} Cost_{(Using\ B,D,E)} &= CC(S)_{(Using\ B,D,E)} + BC(B) + BC(D) + BC(E) \\ &= 240 + 5 + 10 + 20 = 275. \end{aligned} \tag{1.7}$$

In an analogous way, we can calculate the cost of realizing S via D, E, & F (again according to equation 1.3) where we get:

$$\begin{aligned} Cost_{(Using\ D,E,F)} &= (CC(D) \times NoC(D) + CC(E) \times NoC(E) + CC(F) \times NoC(F)) \\ &\quad \times NoC(A) + BC(D) + BC(E) + BC(F) \\ &= (2 \times 15 + 10 \times 7 + 10 \times 8) \times 2 + 10 + 20 + 15 \\ &= (30 + 70 + 80) \times 2 + 45 = 405. \end{aligned} \tag{1.8}$$

The following summarizes all the attained costs to implement S:

$$\begin{aligned} Cost_{(Using\ S)} &= 330 \\ Cost_{(Using\ A)} &= 90. \\ Cost_{(Using\ B,D,E)} &= 275. \\ Cost_{(Using\ D,E,F)} &= 405. \end{aligned}$$

As you can see in this example, there were four options capable of realizing the functionalities of the S block. If for instance a designer seeks a least cost design to implement S, then he will be choosing the A operator to be installed inside the terminal because realizing S using the A operator yielded the minimum cost (90) among all the four possible options of implementation.

1.5 Conclusions

In this chapter, we have traced the evolution of the SDR technology. We further highlighted one approach for designing an SDR system, called the parametrization approach, which consists of software components whose behavior can be changed by reconfiguration procedure. The two approaches of parametrization were explained but more details were exploited in the theoretical approach of parametrization, as it forms the foundation of this thesis subject.

In the theoretical context of parametrization, first an oriented graph that shows the interrelationship between various components of the system was illustrated, which exhibits all the possible options of implementing an SDR multi-standard system. Then, a suggested cost function equation which provides a cost value for any selected alternative was presented and elaborated in details. The goal is to help choose one of the plenty options of implementation, which has the minimum cost, and thus solving the optimization problem that finds balance between flexibility and computing efficiency. Many optimization techniques were previously proposed in this context and will be addressed in following chapters but however, our aim in this thesis is to explore new theoretical tools for solving this optimization problem, particularly by using *Graph Theory*. Hence in the next chapter, we will introduce various fundamental and necessary definitions and applications in graph theory as well as present some basic definitions related to the complexity theory, which will be later exploited to study the complexity of our optimization problem.

Chapter 2

Graph theory and its applications

Contents

Many real-world situations can suitably be described by means of a diagram consisting of a collection of points together with sets combining some of the related points. Graph theory emerged in order to theoretically model such problems and study their properties and characteristics. In chapter one, we showed how the problem of designing an optimal SDR multi-standard system can be represented by means of a graphical structure. However, our aim in this work is to provide a theoretical model of this graphical representation using graph theory.

This chapter is divided into three major parts. The first part, which constitutes the first four sections, introduces some fundamental aspects and theorems related

respectively to graphs and digraphs along with their generalization versions, hypergraphs and directed hypergraphs. In section 5 which forms the second part of this chapter, some basic definitions of the theory of complexity are presented in some depth. In this part, we present different classes of problems, namely the class P, NP, and NP-complete, in order to introduce the notion of polynomially solved problems versus intractable ones. This theory will be later exploited in subsequent chapters to theoretically state and study the complexity of our emerging optimization problem related to the SDR multi-standard design. Finally, section 6 which forms the third and last part of this chapter provides some examples of the numerous realized problems and applications of graph theory, especially to network connection and telecommunication problems, as well as discuss the complexity associated with each.

2.1 Graphs

A *graph* G is an ordered pair $(V(G), E(G))$ consisting of the finite, non-empty set $V(G)$ whose elements are termed *vertices* (or nodes) and the set $E(G)$ of unordered pairs of vertices in $V(G)$. When there is no scope for ambiguity, the letter G is omitted from graph-theoretic symbols, for example we write V and E instead of $V(G)$ and $E(G)$. Each element $e = \{u, v\} \in E(G)$ $(u, v \in V(G))$ is called an *edge* and is said to join the vertices u and v; u and v are called the *ends* of e. For simplicity, the edge $\{u, v\}$ is sometimes written as uv. The ends of an edge are said to be incident with the edge, and vice versa. Two vertices which are incident with a common edge are said to be adjacent, as are two edges which are incident with a common vertex [62].
The number of vertices and edges in G are denoted by $v(G)$ and $e(G)$; these two basic parameters are called the *order* and *size* of G, respectively.

An edge with identical ends is called a *loop*, and an edge with distinct ends is called a *link*. Two or more links with the same pair of ends are said to be *multiple edges*. Fig. 2.1 represents diagrams of several types of graphs. "Dots" are used to represent vertices of a graph and "lines" are used to denote its edges. In Fig. 2.1.b one can find multiple edges with ends v_1 and v_2, which are plot as lines with the same ends, and a loop joining the vertex v_3 to itself. Such a graph G which allows both loops and multiple edges is called a pseudograph. A graph which allows only multiple edges, like the one in Fig. 2.1.a, is called a multigraph. On the contrary, a *simple graph* doesn't accept neither loops nor multiple edges, as the graph of Fig. 2.1.c.

2.1.1 Subgraphs

Let G be a graph. H is said to be a *subgraph* of G if simply $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. In the following, we will present some particular types of subgraphs.

Edge-deleted and Vertex-deleted subgraph let $e \in E(G)$. A subgraph of G of size $e(G) - 1$, called an *edge-deleted subgraph* and denoted by $G \backslash e$, is obtained

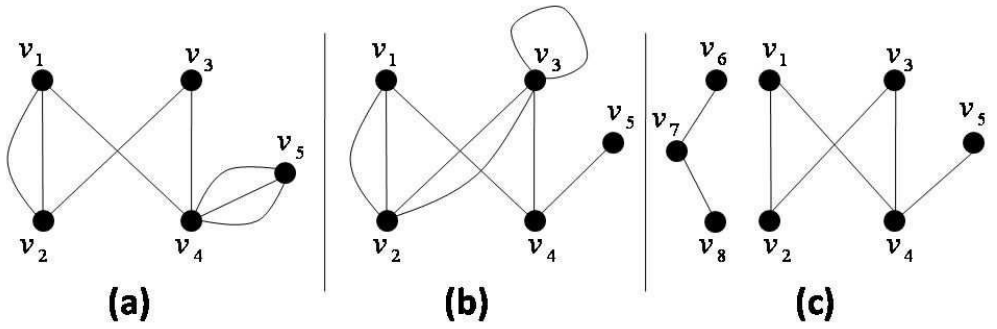

Figure 2.1: Types of graphs: a) multigraph, b) pseudograph and c) simple graph

by deleting e from G but leaving the vertices and the remaining edges intact. Similarly, if v is a vertex of G, we may obtain a subgraph on $v(G) - 1$ vertices, called a *vertex-deleted subgraph* and denoted by $G - v$, by deleting from G the vertex v together will all the edges incident with v. These subgraphs represent respectively the operations of an edge deletion (sometimes called weak edge deletion) and a vertex deletion (sometimes called strong vertex deletion) in a graph G.

Spanning subgraph A subgraph of G obtained by successive edge deletion is called a *spanning subgraph*. In other words, a subgraph whose vertex set is the entire vertex set of G.

Induced subgraph An *induced subgraph* is a subgraph obtained by successive vertex deletions. In other words, if $U \subseteq V(G)$ is the vertex set of the induced subgraph, its edge set will comprise all the edges of $E(G)$ which join vertices in U. Such a subgraph is called a subgraph of G induced by U.

2.1.2 Various definitions and particular graphs

In this subsection, we will present various definitions and notations concerning graphs which play prominent roles in graph theory. Note that we will say graph to mean a simple graph unless otherwise stated.

Independent graph An *independent graph* is one in which no two vertices are adjacent, i.e a graph whose edge set is empty.

Complete graph A *complete graph* is a graph in which any two vertices are adjacent. A complete graph of order n is denoted by K_n. The complete graph K_4 is illustrated in Fig. 2.3.a.

Neighbors and Degrees Let $G = (V, E)$ be a graph (where G is not necessarily a simple graph) and let $v \in V$. A vertex adjacent to v is called a *neighbor* of v. The *neighborhood* of v, denoted by $N_G(v)$, is the set of all neighbors of v.
The *degree* of v in a graph G, denoted by $d_G(v)$, is the number of edges of G incident with v, each loop counting as two edges. In particular, if G is a simple

graph, $d_G(v)$ is the number of neighbors of v in G. When G is clear from the context, we just write $d(v)$. A vertex of degree zero is called an *isolated* vertex. The maximum degree of a vertex in G is denoted by $\Delta(G)$. The following theorem, the handshaking theorem, establishes a fundamental identity relating the degrees of the vertices of a graph and the number of its edges.

Theorem 1. *For any graph* G, $\sum_{v\in G} d_G(v) = 2e(G)$

Proof. The proof is done by induction on $e(G) = m$. The equality holds trivially for $m = 0$. Suppose that the equality holds for m and let G be a graph such that $e(G) = m + 1$. Select an arbitrary edge $e = xy$ of G, with $x \neq y$. The theorem holds for the edge-deleted subgraph $G' = G\backslash e$ whose size is m. So we have, $\sum_{v\in G'} d_{G'}(v) = 2e(G') = 2m$. Then:

$$\begin{aligned}\sum_{v\in G} d_G(v) &= \sum_{v\neq x,y} d_G(v) + d_G(x) + d_G(y)\\ &= \sum_{v\neq x,y} d_{G'}(v) + d_{G'}(x) + d_{G'}(y) + 2\\ &= \sum_{v\in G'} d_{G'}(v) + 2\\ &= 2m + 2 = 2(m + 1) = 2e(G)\end{aligned}$$

In a similar reasoning, the equality still holds when the selected edge $e = xx$ is a loop, since this edge contributes twice to the degree of x. □

Path A *path* is a graph P whose vertices can be arranged in a linear sequence in such a way that two vertices are adjacent if they are consecutive in the sequence, and are nonadjacent otherwise. The number of edges in the path is the *length* of the path, denoted by $l(P)$, which is evidently equal to $v(P) - 1$.

Cycle A *cycle*, on three or more vertices, is a graph whose vertices can be arranged in a cyclic sequence in such a way that two vertices are adjacent if they are consecutive in the sequence and are nonadjacent otherwise. In multigraphs or pseudographs, a cycle on one vertex forms a loop, and a cycle on two vertices consists of a pair of multiple edges. The *length* of a cycle is the number of its edges (and necessarily number of its vertices), denoted by $l(C)$. A cycle is termed *even* if its length is even, and *odd* otherwise.
A graph G is said to be *acyclic* if it contains no cycles; otherwise, it's called *cyclic*.

Walk and Trail A *walk* W in a graph G is a finite alternating sequence of vertices and edges (not necessarily distinct), $W = v_0e_1v_1\cdots v_{p-1}e_pv_p$ such that v_{i-1} and v_i are the ends of e_i, $1 \leq i \leq p$. When there is no ambiguity, the walk is denoted by $W = v_0v_1\cdots v_{p-1}v_p$. The integer p (number of edges) is the *length* of W, denoted by $l(W)$. The vertices v_0 and v_p are called the ends of W, v_0 being its *origin vertex* and v_p its *destination vertex*. A walk is termed *closed* if $v_0 = v_p$, and *open* otherwise. A walk is termed a *trail* if all of its edges are

distinct.
Remark that a path (resp. cycle) is nothing but an open walk (resp. closed walk) whose vertices (and necessarily all edges) are distinct. If $v_0 = x$ and $v_p = y$, we call the walk W (path and trail respectively) an $xy-walk$ ($xy-path$ and $xy-trail$ respectively). If an $xy-path$ exists in a graph G, then we say that x is reachable from y (and vice versa) and that x and y are connected in G. A path, cycle and trail are pictured in Fig 2.2.

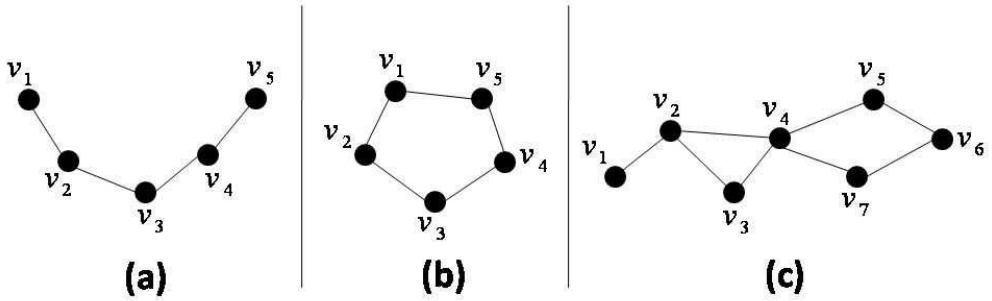

Figure 2.2: a) path of length 4, b) cycle of length 5 and c) trail of length 8

Connection A graph G (not necessarily simple) is said to be *connected* if there exists at least one path between every pair of vertices in G; otherwise, the graph is *disconnected*. Alternatively, we say that a graph G is connected if for every partition of its vertex set into two nonempty sets X and Y, there is an edge with one end in X and one end in Y. In fact, we can prove the equivalence between these two definitions as follows:
Consider a partition X, Y of $V(G)$ $(X, Y \neq \phi)$ and let $u \in X$, $v \in Y$. By hypothesis, there exists at least one path between u and v in G. Such a path contains an edge with one end in X and the other end in Y.
Conversely, let $u, v \in V(G)$. Set $X = \{x \in V(G); G \text{ contains a } ux-\text{path}\}$ and $Y = V(G) \backslash X$. If $v \in Y$ then X, Y forms a partition of $V(G)$ and by hypothesis, $\exists$ an edge $ww' \in E(G)$ with $w \in X$ and $w' \in Y$. Consequently, G contains a uw' path which forms a contradiction. Thus $v \in X$.
A *connected component* of a graph G is a maximal (with respect to inclusion $(\subseteq)$) connected subgraph of G. Obviously, any connected graph contains only one connected component which is the graph itself.
As an example, one can remark that Figures 2.1.a and 2.1.b are connected graphs while Fig. 2.1.c is not connected, containing two connected components; one induced by the set $S_1 = \{v_1, v_2, v_3, v_4, v_5\}$ and the other induced by the set $S_2 = \{v_6, v_7, v_8\}$

Distance The distance between two vertices u and v in a graph G, denoted by $d_G(u, v)$, is the length of a shortest $uv - path$. A shortest path joining u and v is called a $uv - geodesic$. If there is no such path connecting u and v (i.e if u and v lie in two distinct connected components of G), then $d_G(u, v)$ is set to ∞.

Cut Edges An edge e is said to be a *cut edge* of a connected graph G if its deletion results in a disconnected graph, i.e if $G\backslash e$ is a disconnected subgraph. We have the following characterization of cut edges, whose proof is straightforward [62]:

Theorem 2. *An edge e of a graph G is a cut edge if and only if e doesn't belong to any cycle in G.*

Stable set A *stable set* S of a graph G is a subset of $V(G)$ in which no two of its vertices are adjacent in G. In other words, the subgraph of G induced by S is an independent graph.
The largest cardinality of a stable in a graph G is called the *stability number*, denoted by $\alpha(G)$. Simply, $\alpha(G)$ is the maximum number of pairwise nonadjacent vertices.

Bipartite graphs A graph $G = (V, E)$ is called *n-partite*, $n > 1$, if V can be partitioned into n stables, $V_1, V_2, \cdots, V_n$ called its *parts*; such that every edge in E joins a vertex of V_i to a vertex of V_j, with $i \neq j$. When $n = 2$, the n-partite graph is called a *bipartite graph*. A characterization of a bipartite graph was possible. In fact, König proved in 1928 that a graph is bipartite if and only if it contains no odd cycles [62].

When the vertex set of a bipartite graph G is partitioned into the two parts X and Y in which every vertex in X is joined to every vertex in Y, then G is called a *complete bipartite graph*. If $|X| = r$ and $|Y| = s$, then G will be denoted by $K_{r,s}$. The number of edges in $K_{r,s}$ clearly equals rs. A *star* is a complete bipartite graph with $|X| = 1$ or $|Y| = 1$. Figues 2.3.a and 2.3.b and 2.3.c show diagrams of the complete graph of order 4 (K_4), the complete bipartite graph $K_{2,3}$, and the star $K_{1,5}$ respectively.

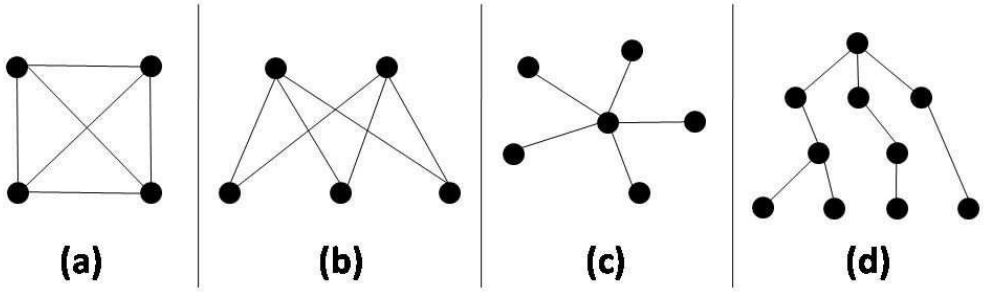

Figure 2.3: a) The complete graph K_4, b) the complete bipartite graph $K_{2,3}$, c) the star $K_{1,5}$, and d) a tree

Trees and Forests A connected acyclic graph is called a *tree*. In an acyclic graph, each connected component is a tree. For this reason, acyclic graphs are usually called *forests*. It's possible to specify a number of alternative definitions of a tree proved all to be equivalent:

- A tree is a graph in which any pair of its vertices is connected by a unique path.

- A tree is a connected graph G with $\upsilon(G) = e(G) + 1$.
- A tree is an acyclic graph G with $\upsilon(G) = e(G) + 1$.
- A tree is an acyclic graph which has the property that if any two of its vertices which are not adjacent are joined directly by an edge then the resulting graph possesses exactly one cycle.

An example of a tree is illustrated in Fig. 2.3.d.
A *rooted tree* $T(r)$ is a tree T with a specified vertex r, called the *root* of T.
A *subtree* of a graph G is a subgraph which is a tree. If this tree is a spanning subgraph, it is called a *spanning tree* of G. A characterization of connected graphs by means of spanning trees is stated as follows: *a* graph is connected if and only if it contains a spanning tree. In fact, if a graph G has a spanning tree T, then any two vertices of G are connected by a path in T, and hence in G. Conversely, if G is connected but is not a tree and e is an edge of a cycle in G, then $G\backslash e$ is a spanning subgraph of G which is also connected because, by theorem 2, e is not a cut edge of G. By repeating this process of deleting edges in cycles until every edge which remains is a cut edge, we obtain a spanning tree of G.

Hamiltonian graphs A graph G is said to be *Hamiltonian* if it contains a cycle passing by all the vertices of G, i.e if it contains a spanning subgraph which is a cycle. Such a cycle, when it exists, is called a *Hamiltonian cycle.*
Many mathematicians over the decades have unsuccessfully attempted to find an elegant characterization of Hamiltonicity. However, sufficient conditions for a graph to be Hamiltonian were attained. We start by the following theorem (whose proof can be found in [61]) before discussing some of these sufficient conditions.

Theorem 3. *Consider a connected graph G of order n ($n > 2$) and let u and v be a pair of distinct nonadjacent vertices of G such that $d(u) + d(v) \geq n$. Then $G + uv$ is Hamiltonian if and only if G is Hamiltonian.*

The concept discussed in this theorem of adding an edge leads to a useful definition concerning Hamiltonicity.

Definition 1. The closure of a graph G of order n, denoted by $C(G)$, is the graph obtained from G by successively joining pairs of nonadjacent vertices whose degree sum is at least n (in the graph obtained at each step of joining), until it is not possible to join any further pairs [61].

A closure operation example is illustrated in Fig 2.4 The next theorem is required to establish a sufficient condition for Hamiltonicity, whose proof is a direct result of theorem 3 and the definition of closure.

Theorem 4. *A graph is Hamiltonian if and only if its closure is Hamiltonian.*

Theorem 5. *Let G be a graph with at least three vertices. If $C(G)$ is complete, then G is a Hamiltonian graph.*

Proof. The proof of this theorem is immediate by theorem 4 and the fact that each complete graph with at least 3 vertices is Hamiltonian. □

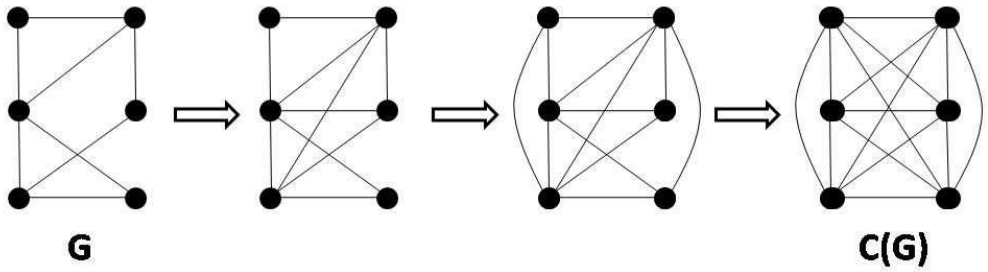

Figure 2.4: A complete closure operation

Many corollaries rise from theorem 5 which also provide sufficient conditions for Hamiltonicity, although are weaker than theorem 5 itself. We mention only two of them.

Corollary 1. *If $G = (V, E)$ is a graph of order n, where $n \geq 3$, such that any two distinct nonadjacent vertices u and v in V satify that $d(u) + d(v) \geq n$, then G is Hamiltonian (Ore's theorem).*

Corollary 2. *If $G = (V, E)$ is a graph of order n ($n \geq 3$) such that $d(v) \geq \frac{n}{2}$ for every vertex v of G, then G is Hamiltonian (Dirac's theorem).*

Weighted Graphs and Subgraphs A graph $G = (V, E)$ (not necessarily simple) is termed *weighted* if there exists a function $w : E \to \mathbb{R}$ which assigns a real number, called *weight*, to each edge of E. The weighted graph G is denoted by (G, w). In some very particular cases, it would be necessary to assign a real weight vector to each edge.
Usually, the weight of an edge represents how unfavorable it is. Sometimes, the word cost, or length, or capacity is used instead of weight. Many real-world situations ultimately reduce to some kind of weighted graph problems. In a communication network, for example, one might consider the cost of transmitting data along a link or of constructing a new link between communication centres.

If $F = (V(F), E(F))$ is a subgraph of a weighted graph, the *weight* $w(F)$ of F is defined to be the sum of the weights on its edges, $\sum_{e \in E(F)} w(e)$. Many optimization problems amount to finding, in a weighted graph, a subgraph of a certain type with minimum or maximum weight, whether a tree or a path or etc $\cdots$. Note that the distance between two nodes u and v in a weighted graph (G, w), also denoted by $d_G(u, v)$, is the *minimum weight* of a $uv - path$.

Graph coloring and Chromatic number A graph G is said to be m-colorable if the vertices of G can be colored by means of m colors in such a way that 2 adjacent vertices have 2 distinct colors. In other words, $V(G)$ can be partitioned into m stables $S_1, S_2, \cdots, S_m$, i.e $V(G) = S_1 \cup S_2 \cup \cdots \cup S_m$ with $S_i \cap S_j = \phi$ $\forall \quad i \neq j$. The *chromatic number* $\chi(G)$ is by definition the minimal number m such that G is m-colorable. For example, the chromatic number of a bipartite

graph is 2, that of an odd cycle is 3, and that of K_n is n.

Let G be a graph. Then we have $\alpha(G)\chi(G) \geq \upsilon(G)$. This is straightforward because $V(G)$ can be partitioned into $\chi(G)$ stables S_i, with $|S_i| \leq \alpha(G)\ \forall\ i$.
It has been proved that $\chi(G) \leq \Delta(G) + 1$, and Brooks further proved in 1941 that if G is indeed neither complete nor an odd cycle, then $\chi(G) \leq \Delta(G)$ [93].

2.2 Digraphs

In graphs, edges were unordered pairs of vertices of V. Sometimes, it's useful to give each edge an orientation or a direction. When dealing with problems of traffic flow, for example, it is necessary to know which roads in the network are one-way, and in which direction traffic is permitted. In this context, a graph in which each link has an assigned orientation is introduced.

A *directed graph* D (or *digraph* for short) is defined to be an ordered pair of sets (V, A), where V is a finite, non-empty set of vertices and A is a set of ordered pairs of (not necessarily distinct) vertices of V. The elements of A are called arcs. If $a = (u, v)$ is an arc in A, then a is said to *join* u to v; one also says that u *dominates* v or v is *dominated* by u. The vertex u is the *tail* of a while v is its head; they are the two *ends* of a. The number of arcs in D is denoted by $a(D)$. Note that the definition of a digraph doesn't avoid oppositely directed pair of arcs joining the same pair of vertices.
An arc of the form (v, v), $v \in V$ is termed a loop. Parallel arcs are ones with the same tail and the same head. In a similar manner to graphs, a *simple digraph* is one with no loops or parallel arcs. A digraph which relaxes the constraint in the definition that no parallel arcs are allowed is called a *multidigraph*, while that which in addition accepts loops is called a *pseudodigraph*.

With any digraph D, one can associate a graph G by just replacing each of its arcs by an edge with the same ends (i.e by ignoring the orientation of the arcs); this graph is called the *underlying graph* of D, denoted by $G(D)$. Conversely, one may obtain a digraph from a graph G by replacing each edge by just one of the two possible arcs with the same ends. Such a digraph is called an *orientation* of G, denoted by $\vec{G}$. An orientation of a simple graph is referred to as an *oriented graph*. One particular interesting orientation is that of a complete graph. Such an oriented graph is called a *tournament*. A *bipartite digraph* is an orientation of a bipartite graph.
Various variants of digraphs are illustrated in Fig. 2.5.

2.2.1 Different interesting definitions and types of digraphs

As in graphs, various appealing definitions and theorems can also be associated to digraphs. In this subsection, we will mention some of the most important of these aspects.

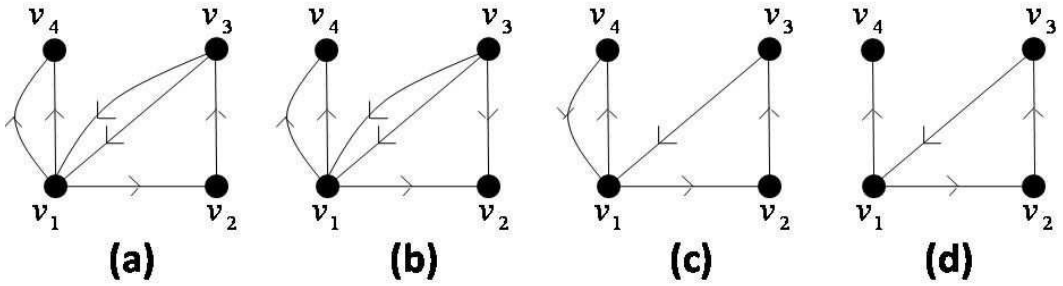

Figure 2.5: Various types of digraphs: a) multidigraph, b)pseudodigraph, c) simple digraph, and d) oriented graph

Subdigraph A digraph $D' = (V', A')$ is a *subdigraph* of a digraph $D = (V, A)$ if $V' \subseteq V$ and $A' \subseteq A$.

Neighbors and degrees in digraphs Let $D = (V, A)$ be a digraph and $u \in V$. The vertices which dominate a vertex u in a digraph D are its *in-neighbors*, those which are dominated by the vertex u are its *out-neighbors*. These sets are denoted by $N_D^-(u)$ and $N_D^+(u)$ respectively.
The *indegree* $d_D^-(u)$ of vertex u in D is the number of arcs with head u, and the *outdegree* $d_D^+(u)$ of u in D is the number of arcs with tail u (loops counting twice). In particular, $d_D^-(v) = |N_D^-(v)|$ and $d_D^+(v) = |N_D^+(v)|$ in simple digraphs and oriented graphs. As usual, the index D can be omitted if this evidently doesn't lead to misunderstanding. The maximum indegree and outdegree of D are denoted by $\Delta^-(D)$ and $\Delta^+(D)$ respectively.

The *degree* of a vertex u in a digraph D is simply the degree of u in $G(D)$. In other words, the degree of u in D is the sum of its indegree and outdegree in D.

The handshaking theorem stated for graphs can also be stated for digraphs in terms of the indegrees and outdegrees of vertices as follows:

Theorem 6. *Let $D = (V, A)$ be a digraph. Then*

$$\sum_{v \in D} d_D^+(v) = \sum_{v \in D} d_D^-(v) = e(D) = \frac{1}{2} \sum_{v \in D} d_D(v)$$

.

The proof can also be done by induction on $e(D)$ just in a similar way to that used in graphs, but here considering the outdegrees and indegrees of the vertices instead of the degrees.

Source and Sink A vertex of indegree zero is called a *source*. A *sink*, on the contrary, is a vertex whose outdegree is zero.

Directed walks, trails, paths and cycles Let $D = (V, A)$ be a digraph. A *directed walk* W in D is an alternating sequence of vertices and arcs of D (not

necessarily distinct) $W = v_0a_1v_1 \cdots v_{l-1}a_lv_l$ such that v_{i-1} and v_i are the tail and head of a_i respectively, $1 \leq i \leq l$. A directed walk is termed a *directed trail* if all of its arcs are distinct, a *directed path* if all of its vertices are distinct, and a *directed cycle* or *circuit* if all its vertices are distinct as well, except for the fact that $v_0 = v_l$. If x and y are the origin and destination vertices of W, the directed walk W (respectively directed trail, directed path) is called an (x, y)-*walk* (respectively (x, y)-*trail*, (x, y)-*path*). If $x, y \in V$ and there exists an (x, y)-path, then y is said to be *reachable* from x.

Semiwalk, Semitrail, Semipath and Semicycle A *semiwalk* [61] W is an alternating sequence of vertices and arcs (not necessarily distinct)
$W = v_0a_1v_1 \cdots v_{l-1}a_lv_l$ where either one of v_{i-1} or v_i is the tail of a_i while the other is its head, $\forall \quad 1 \leq i \leq l$. A semiwalk is termed a *semitrail* if all of its arcs are distinct, a *semipath* if all of its vertices are distinct, and a *semicycle* if also all of its vertices are distinct except that we have $v_0 = v_l$. Fig. 2.6 presents examples of a directed path, directed cycle, semipath, and semicycle.

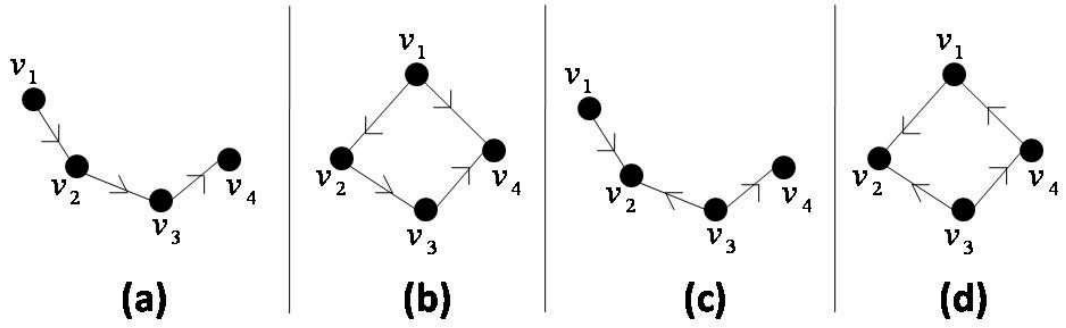

Figure 2.6: a) directed path, b) directed cycle or circuit, c) semipath, and d) semicycle

Connection in Digraphs Many types of connections appear in digraphs; we will define a digraph as being strongly connected, unilaterally connected, or connected as follows: A digraph $D = (V, A)$ is said to be *strongly connected* if every two of its distinct vertices, say u and v, are such that u is reachable from v and v is reachable from u, i.e $\forall \quad u, v \in V$, $\exists$ a (u, v) path. By contrast, D is *unilaterally connected* if either u is reachable from v or v is reachable from u. However, we say that a digraph is *connected* (some books use the term *weakly connected*) if any two distinct vertices of V are joined by a semipath. This is equivalent to saying that the underlying graph of D, $G(D)$, is a connected graph. Clearly, we have:
D is strongly connected $\Rightarrow$ D is unilaterally connected $\Rightarrow$ D is connected.
For example, one can easily notice that the digraph of Fig. 2.5.c is strongly connected, while that of Fig. 2.5.d is not because we can't find any (v_4, v_1) path. However, Fig. 2.5.d represents a unilaterally connected digraph, unlike the connected digraph of Fig. 2.5.b which doesn't contain neither a (v_2, v_4) path, nor a (v_4, v_2) path. Finally, remark that Fig. 2.5.a is not connected in any type of connection in digraphs.

Components Just as there are three concepts of connectivity in digraphs, there are also three kinds of components. A *strong component* (respectively *unilateral component*, *connected component*) in a digraph D is a maximal (with respect to inclusion ($\subseteq$)) strongly connected (respectively unilaterally connected, connected) subdigraph of D.

Out-branching and in-branching An out-branching T is an orientation of a rooted tree, called oriented tree, in which all the vertices have indegree 1 unless exactly the root vertex which is a source. Similarly, an in-branching is an oriented tree, where the outdegree of all its vertices is 1 except for the root vertex which is a sink. Fig. 2.7 provides a pictorial view of an out-branching and in-branching.

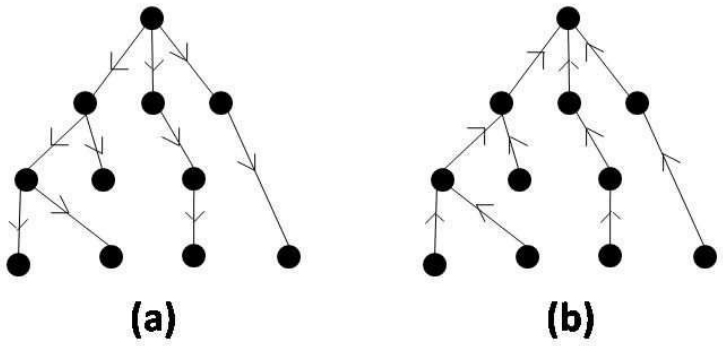

Figure 2.7: a) An out-branching and b) an in-branching

Weighted digraphs As for weighted graphs, a *weighted digraph* is a couple (D, w) where $D = (V, A)$ is a digraph and $w : A \to \mathbb{R}$ is a function which assigns real number weights to each arc of D.

2.3 Hypergraphs

The basic idea of the hypergraph concept is to consider a generalization of a graph in which any subset of a given set may be an edge rather than two-element subsets. A *hypergraph* H in this context (sometimes called a *set-system*) is a pair $(V(H), E(H))$ of sets, with $V(H)$ is a non-empty set of *vertices* and $E(H)$ is a finite set of subsets of $V(H)$ (not necessarily non-empty) called *hyperedges*. Some hyperedges may be subsets of some others, in which case they are called *included*. Other hyperedges may coincide; this is when they are called *multiple hyperedges*. A hyperedge of cardinality 1 is called a *loop*.
A *simple hypergraph* (also known as *Sperner families*) is a hypergraph which contains no included edges, and thus has no empty or multiple hyperedges [68]. A simple graph is a simple hypergraph each of whose hyperedges is of cardinality 2.

Let $H = (V(H), E(H))$ be a hypergraph. The *order* of a hypergraph is the number of its vertices, denoted by $n(H)$, while the number of hyperedges in H is denoted by

$m(H)$. Two vertices are said to be *adjacent* in H if there is a hyperedge in $E(H)$ that contains both vertices, and two hyperedges of H are *adjacent* if their intersection is not empty. If a vertex $x \in V(H)$ belongs to a hyperedge $E \in E(H)$, then they are said to be *incident* to each other. It's always the case in graph theory where the notion of adjacency is referred to the elements of the same kind (vertices vs vertices, or edges vs edges), while the incidence is referred to the elements of different kind (vertices vs edges).

Hypergraphs can model concepts in different sciences in a much more general setting than graphs do. In addition, they help to find optimal solutions for many new optimization problems. Example, a hypergraph modeling in computer science is one in which the vertices are computers in a network and the hyperedges are the subsets of computers with devices from different manufacturers, one subset for every manufacturer; another is in broadcasting where the vertices are radio transmitters in a region and the hyperedges are the subsets of transmitters which transmit on the same frequency right now, one subset for each frequency; in healthcare for example, the vertices can represent illnesses and the hyperedges will be the subsets of illnesses which can be treated by some medicines, one hyperedge for each medicine, just to mention few.

A hypergraph $H = (V(H), E(H))$ is given another pictorial representation as a bipartite graph $B(H) = (V_1(B) \cup V_2(B), E(B))$, called the *bipartite representation graph* [66], as follows:

- $V_1(B) = V(H), \quad V_2(B) = E(H)$.
- An edge in $B(H)$ between $x \in V_1(B)$ and $E \in V_2(B)$ is drawn if and only if $x \in E$ in H.

Fig. 2.8 is an example of a hypergraph $H = (V(H), E(H))$ together with its bipartite representation $B(H)$ where:

$$V(H) = \{x_1, x_2, x_3, x_4, x_5, x_6\},\ E(H) = \{E_1, E_2, E_3, E_4\},$$
$$E_1 = \{x_1\},\ \ E_2 = \{x_1, x_2\},\ \ E_3 = \{x_1, x_2, x_4\},\ \ E_4 = \{x_2, x_3, x_5\}$$
$$\text{AND}$$
$$V_1(B) = \{x_1, x_2, x_3, x_4, x_5, x_6\} \text{ and } V_2(B) = \{E_1, E_2, E_3, E_4\}.$$
$$E(B) = \{x_1E_1, x_1E_2, x_1E_3, x_2E_2, x_2E_3, x_2E_4, x_3E_4, x_4E_3, x_5E_4\}$$

2.3.1 Subhypergraphs

In this subsection, we will introduce two basic hypergraph operations which allow to obtain one hypergraph from another and to create various types of subhypergraphs [67]. Let $H = (V(H), E(H))$ be a hypergraph.

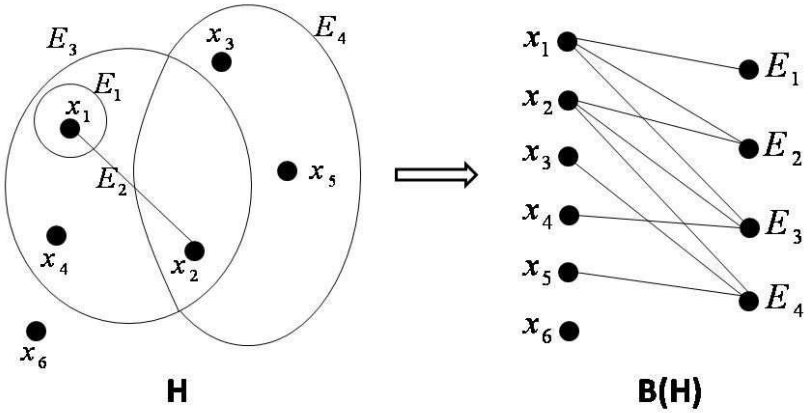

Figure 2.8: Example of a hypergraph H and its bipartite representation graph $B(H)$

vertex deletion Let $x \in V(H)$. A *deletion* of x (sometimes called strong vertex deletion) from H is obtained by removing all the hyperedges containing x from $E(H)$ and removing of x from $V(H)$. The obtained hypergraph is denoted by $H - x$.

hyperedge deletion This is the simplest operation of deletion, where a hyperedge E is just removed from the list of hyperedges $E(H)$ (sometimes called weak hyperedge deletion). Such edge-deleted hypergraph is denoted by $H \backslash E$.

A *subhypergraph* of H is any hypergraph $H' = (X', D')$ such that $X' \subseteq V(H)$ and $D' \subseteq E(H)$. Evidently, H' can be obtained from H by deleting the vertices from the set $V(H) \backslash X'$ and further hyperedge deletion of the remaining hyperedges from $E(H) \backslash D'$. Accordingly $H - x$ and $H \backslash E$ are subhypergraphs of H. The following includes two particular types of subhypergraphs.

Induced subhypergraphs A hypergraph $H' = (X', D')$ is called an *induced subhypergraph* of a hypergraph $H = (X, D)$ if $X' \subseteq X$ and all hyperedges of H which are subsets of X' form the family D'. In other words, X' is obtained from H by the vertex deletion of all the vertices in $V(H) \backslash X'$. We say that H' is a subhypergraph of H induced by X'.

Partial or Spanning subhypergraphs For a hypergraph $H = (X, D)$, any subhypergraph H' such that $H' = (X, D')$ is called a *partial* or *spanning subhypergraph*; i.e a spanning subhypergraph of H has the same vertex set as that of H and is obtained by just deleting the hyperedges in $D \backslash D'$.

2.3.2 Basic definitions and particular cases concerning hypergraphs

Let $H = (V(H), E(H))$ be a hypergraph and $x \in V(H)$. Some basic definitions and notations of hypergraphs will be presented hereby.

Empty hypergraph An *empty hypergraph* H is one whose hyperedge set is empty, i.e $E(H) = \phi$.

Neighbor and Neighborhood A vertex adjacent to x in H is called a *neighbor* of x. The *neighborhood* of x, denoted by $N(x)$, is the set containing all neighbors of x.

Star with center x The star $S(x)$ with center x is the set of hyperedges containing x, i.e $S(x) = \{E \in E(H), x \in E\}$.

Vertex degree The *degree* of x in H, denoted by $d_H(x)$, is the number of hyperedges in $E(H)$ incident with x; i.e $d_H(x) = |S(x)|$. If x is incident to no hyperedge in H, i.e $d_H(x) = 0$, it's called an *isolated* vertex. The *maximum degree* of a vertex in $V(H)$ is denoted by $\Delta(H)$. The following theorem generalizes the handshaking theorem to hypergraphs which establishes an equality between the sum of all vertex degrees and the sum of all edge cardinalities [67].

Theorem 7. *For a hypergraph H, $\sum_{x\in V(H)} d_H(x) = \sum_{E\in E(H)} |E|$.*

Proof. Consider the bipartite represetation graph of H, $B(H) = (V_1(B) \cup V_2(B), E(B))$. We have $\sum_{x\in V_1(B)} d_{B(H)}(x) = \sum_{x\in X} d_H(x)$ and $\sum_{x\in V_2(B)} d_{B(H)}(x) = \sum_{E\in E(H)} |E|$. Evidently $\sum_{x\in V_1(B)} d_{B(H)}(x) = \sum_{x\in V_2(B)} d_{B(H)}(x)$, because both sums represent the number of edges in $B(H)$. □

Hyperedge size Let $E \in E(H)$. The number $|E|$ is called the *size* of the hyperedge E. The *rank* of a hypergraph H is the maximum size of a hyperedge in $E(H)$, denoted by $r(H)$. A hypergraph in which all hyperedges have the same size $r \geq 0$ is called an *r-uniform* hypergraph. Thus, a simple graph is a 2-uniform hypergraph.

Complete hypergraphs A simple hypergraph H of order n is called a *complete r-uniform hypergraph*, denoted by K_n^r, when $E(H)$ coincides with all the r-subsets of X. Thus a complete graph on n vertices is a complete 2-uniform hypergraph K_n^2.

Walk, trail, path and cycle in Hypergraphs In a hypergraph H, a sequence of nodes and hyperedges $W = x_0E_1x_1 \cdots x_{l-1}E_lx_l$ satisfying $x_{i-1}, x_i \in E_i$, $1 \leq i \leq l$ is called a *walk* connecting the vertices x_0 and x_l, or, equivalently, *x_0x_l-walk*; it's called an *x_0x_l-trail* if all hyperedges are distinct. Indeed, if all nodes $x_0, x_1, \cdots, x_l$ are distinct, it will be called an *x_0x_l-path*. If in addition we have $x_l \in E_1$, then W is called a cycle (denoted *x_0x_l-cycle*); this is in particular true when $x_0 = x_l$. The value l denotes the *length* of the walk, trail, path or cycle.

Connection in hypergraphs As in graphs, a hypergraph H is called *connected* if for any pair of its vertices, there is a path connecting them. A *connected component* of H is a maximal connected suhypergraph (w.r.t $\subseteq$) of H.

Stable sets in hypergraphs [67] A set $S \subseteq V(H)$ is said to be a *stable* if it doesn't contain any hyperedge E with $|E| > 1$. The largest size of a stable set over all maximal by inclusion stable sets is called the *stability number*, denoted by $\alpha(H)$.

Bipartite hypergraphs [67] A hypergraph H is called *bipartite* if its vertex set can be partitioned into two disjoint sets X_1 and X_2 (called *parts*) in such a way that each hyperedge of cardinality ≥ 2 contains vertices from both parts, meaning that there is no such hyperedge completely inside X_1 or X_2.

Weighted hypergraphs [69] A *weighted hypergraph* is one in which each hyperedge E is assigned a real weight vector $w(E)$, i.e there exists a function w which maps each hyperedge of H to a real vector. Depending on the particular application, the components of $w(E)$ may represent *costs, lengths, capacities,* etc. For the sake of simplicity, only scalar weights are usually considered.

Hypergraph coloring and Chromatic number [68] Let k be an integer ≥ 2. A *k-coloring* of the vertices of H is a partition $(S_1, S_2, \cdots, S_k)$ of the set of vertices $V(H)$ into k stables such that every hyperedge which is not a loop has intersections with at least two classes of the partition; i.e $E \in E(H)$ s.t $|E| > 1$ $\Rightarrow$ E isn't a subset of $S_i \quad \forall \quad i = 1, 2, \cdots, k$. A vertex in S_i is said to be a "vertex of color i" and S_i is called "the color set i", where S_i may possibly be empty; the only monochromatic edges are therefore the loops.
The *chromatic number* of H, denoted $\chi(H)$, is the smallest integer k such that H admits a k-coloring.
For example, if H is the hypergraph whose vertices are the different waste products in a chemical production factory, and in which the hyperedges are the dangerous combinations of these waste products, the chromatic number of H will represent the smallest number of waste disposal sites that the factory needs in order to avoid any hazardous situation.

If H is a hypergraph of order n with stability number $\alpha(H)$, then $\alpha(H)\chi(H) \geq n$ (exactly as in graphs).

Mixed hypergraph and its proper coloring [67] A hypergraph $H = (V(H), E(H))$ is called a *mixed hypergraph* if its set of hyperedges $E(H)$ is partitioned into two sets C and D (also denoted by $C(H)$ and $D(H)$ respectively), where the elements in C are called C-hyperedges and those in D are called D-hyperedges. In this case, the hypergraph H will be denoted by the triple
$H = (V(H), C(H), D(H))$ or simply $H = (V, C, D)$. This splitting manner is useful in many cases when some hyperedges in a hypergraph have common aspects (or are required to have) and the rest have something else in common. A new beneficial notion of coloring was introduced, called *proper coloring* of mixed hypergraphs.

A *proper coloring* of a mixed hypergraph $H = (V, C, D)$ is a mapping $c : V \to \{1, 2, \cdots \lambda\}$ satisfying the following two conditions:

1. every C-hyperedge has at least two vertices of a common color.
2. every D-hyperedge has at least two vertices of different colors.

Definition 3. A mixed hypergraph H is called *colorable* if it admits at least one proper coloring; otherwise it's called uncolorable.

In graphs and hypergraphs, we can always find a coloring. On the contrary, it is easy to remark that not every mixed hypergraph is colorable [67].

2.4 Directed Hypergraphs

In hypergraphs, a hyperedge was a subset of the vertex set. However, in many applications, it might be sometimes useful to consider some of the elements of a hyperedge directed or oriented towards others. A *directed hyperedge*, in this context, is an ordered pair (X, Y) of disjoint subsets of the vertex set in which the elements of X are directed towards the elements in Y. A *directed hypergraph* is a hypergraph but with directed hyperedges.

Formally, a directed hypergraph H is a pair $(V(H), E(H))$, where $V(H)$ is a non-empty set of elements (called vertices or nodes) and $E(H)$ is a set of ordered pairs of disjoint subsets of $V(H)$ (called directed hyperedges or *hyperarcs*) [69]. If $E = (X, Y)$ is a hyperarc in $E(H)$, then X & Y are called the tail set and head set of E respectively, denoted by $T(E)$ and $H(E)$.
The hyperarc E is said to *join* the vertices of $T(E)$ to the vertices of $H(E)$. The vertices of $T(E)$ are said to be *incident to* the hyperarc E and the vertices of $H(E)$ are said to be *incident from* E. The vertices of $T(E)$ are *adjacent to* the vertices of $H(E)$, and the vertices of $H(E)$ are *adjacent from* the vertices of $T(E)$.
As in hypergraphs, $n(H)$ and $m(H)$ denote the number of vertices and number of hyperarcs of H respectively.
It's not necessary to present neither the definitions of multiple hyperarcs, loops, $\cdots$ nor the different types of directed hypergraphs, as they can be easily deduced in exactly the same way we have moved from graphs to digraphs.

In a similar manner to hypergraphs, a directed hypergraph $H = (V(H), E(H))$ can be represented by a bipartite digraph $R(H) = (V_1(R) \cup V_2(R), E(R))$ called the *bipartite representation digraph* of H [65] such that:

- $V_1(R) = V(H), \quad V_2(R) = E(H)$
  1. an arc in $E(R)$ is drawn directed from $x \in V_1(R)$ to $E \in V_2(R)$ if and only if $x \in T(E)$ in H.
  2. an arc in $E(R)$ is drawn directed from $E \in V_2(R)$ to $x \in V_1(R)$ if and only if $x \in H(E)$ in H.

Fig. 2.9 represents an example of a directed hypergraph $H = (V(H), E(H))$ along with its bipartite representation $R(H) = (V_1(R) \cup V_2(R), E(R))$ where:

$$V(H) = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\},\ E(H) = \{E_1, E_2, E_3, E_4, E_5, E_6\},$$

$$E_1 = (\{x_1, x_2\}, \{x_3\}),\ \ E_2 = (\{x_3\}, \{x_7\}),\ \ E_3 = (\{x_3, x_4\}, \{x_5, x_6\}),$$
$$\mathrm{E}_4 = (\{x_7\}, \{x_1\}),\ \ E_5 = (\{x_1, x_7\}, \{x_8\}),\ \ E_6 = (\{x_9\}, \{x_8\});$$

AND

$$V_1(R) = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\} \text{ and } V_2(R) = \{E_1, E_2, E_3, E_4, E_5, E_6\}.$$

$$E(R) =$$
$$\{(x_1, E_1), (x_1, E_5), (E_4, x_1), (x_2, E_1), (x_3, E_2), (x_3, E_3), (E_1, x_3), (x_4, E_3), (E_3, x_5), (E_3, x_6),$$
$$(\mathrm{x}_7, E_4), (x_7, E_5), (E_2, x_7), (E_5, x_8), (E_6, x_8), (x_9, E_6)\}$$

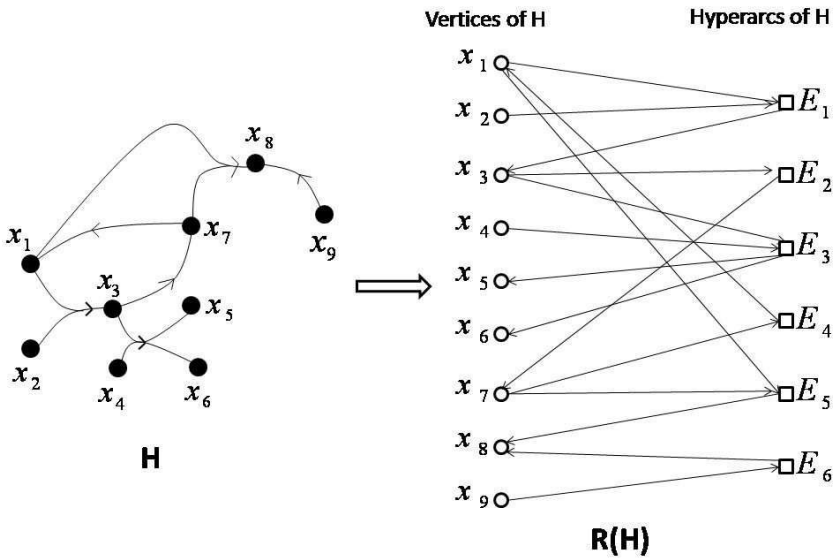

Figure 2.9: Example of a directed hypergraph H and its bipartite representation digraph $R(H)$

2.4.1 Important directed hypergraphs' definitions and notations

Let $H = (V(H), E(H))$ be a directed hypergraph and $v \in V(H)$. Following consists of several important and necessary definitions and aspects of directed hypergraphs, which will represent the keystone of our research work in this thesis.

Subdirected-hypergraph A directed hypergraph $H' = (V(H'), E(H'))$ is called a *subdirected hypergraph* of H if simply $V(H') \subseteq V(H)$ and $E(H') \subseteq E(H)$.

B-hypergraph and F-hypergraph [69] A *Backward* hyperarc, or simply *B-arc*, is a hyperarc $E = (T(E), H(E))$ with $|H(E)| = 1$; a *Forward* hyperarc, or simply *F-arc*, is a hyperarc E such that $|T(E)| = 1$.
A *B-hypergraph* (or simply *B-graph*) is a directed hypergraph whose hyperarcs are B-arcs; Similarly, a *F-hypergraph* (or simply *F-graph*) is a directed hypergraph of which all the hyperarcs are F-arcs.

BF-reductions of hyperarcs [69] Let $E = (T(E), H(E))$ be a hyperarc in H. A BF-reduction of E is a hyperarc $(\{x\}, \{y\})$ where $x \in T(E)$ and $y \in H(E)$. For instance, we have $(\{x_1\}, \{x_3\})$ and $(\{x_2\}, \{x_3\})$ are BF-reductions of the hyperarc E_1 in Fig. 2.9.

Forward and Backward star [69] The Forward Star and the Backward Star of node v are respectively defined by:
$FS(v) = \{E \in E(H), v \in T(E)\}$ and $BS(v) = \{E \in E(H), v \in H(E)\}$.
If $X = (V(X), E(X))$ is a subdirected-hypergraph of a directed hypergraph H, then the forward and backward stars in X of node v ($v \in V(X)$) will be defined as:
$FS_X(v) = \{E \in E(X); v \in T(E)\}$ and $BS_X(v) = \{E \in E(X); v \in H(E)\}$.

Hyperarc sizes [65] If E is a hyperarc in a directed hypergraph H, then $|T(E)|$ is the *in-size* of E while $|H(E)|$ is its *out-size*. The *maximum in-size* and the *maximum out-size* of H are denoted by $s^-(H)$ and $s^+(H)$ respectively. Note that a digraph is a directed hypergraph $D = (V(D), E(D))$ with $s^-(D) = s^+(D) = 1$.

Neighbor and Neighborhood The vertices adjacent from v in a directed hypergraph H are the out-neighbors of v, those which are adjacent to v are its in-neighbors. The set of in-neighbors and out-neighbors of v are denoted by $N_D^-(v)$ and $N_D^+(v)$ respectively.

Vertex degrees The *in-degree* of v, denoted by $d_H^-(v)$, stands for the number of hyperarcs that contain v in their head set, while the *out-degree* of v is the number of hyperarcs that contain v in their tail set, denoted by $d_H^+(v)$. Consequently, we can write: $|BS(v)| = d_H^-(v)$ and $|FS(v)| = d_H^+(v)$. (When H is clear from the context, we will simply write $d^-(v)$ and $d^+(v)$ respectively). The *maximum in-degree* and the *maximum out-degree* of H are $\Delta^-(H)$ and $\Delta^+(H)$ respectively. The next theorem is the handshaking theorem's version extended to directed hypergraphs, which builds equations between the degrees of the vertices and the hyperarcs' sizes.

Theorem 8. *For a directed hypergraph $H = (V(H), E(H))$, the sum of the out-degrees of all the vertices in $V(H)$ is equal to the sum of the in-sizes of all the hyperarcs in $E(H)$, i.e $\sum_{x \in V(H)} d_H^+(x) = \sum_{E \in E(H)} |T(E)|$.*
Similarly, we have $\sum_{x \in V(H)} d_H^-(x) = \sum_{E \in E(H)} |H(E)|$.

Proof. Consider the bipartite representation digraph
$R(H) = (V_1(R) \cup V_2(R), E(R))$ of the directed hypergraph $H = (V(H), E(H))$.
According to the bipartite digraph definition $R(H)$, we can remark that:
$\sum_{x \in V(H)} d_H^+(x) = \sum_{x \in V_1(R)} d_{R(H)}^+(x) = \sum_{E \in V_2(R)} d_{R(H)}^-(E) = \sum_{E \in E(H)} |T(E)|$.
Similarly, we have : $\sum_{x \in V(H)} d_H^-(x) = \sum_{x \in V_1(R)} d_{R(H)}^-(x) = \sum_{E \in V_2(R)} d_{R(H)}^+(E)$
$= \sum_{E \in E(H)} |H(E)|$. □

Directed walk, trail, path and cycle in directed hypergraphs [69] A *directed walk* from vertex r to vertex n in a directed hypergraph $H = (V, E)$ is an alternating sequence of nodes and hyperarcs $P = (v_1 = r, E_{i_1}, v_2, E_{i_2}, v_3, \cdots, E_{i_q}, v_{q+1} = n)$ where: vertex $v_j \in T(E_{i_j})$ and $v_{j+1} \in H(E_{i_j})$ for each $1 \leq j \leq q$. Such a walk is called an (r, n)-*walk*, where r is its origin and n is its destination. If all hyperarcs are distinct, P will be a *directed trail* (called (r, n)-*trail*); if in addition all the vertices were distinct, then P is called a *directed path* from r to n (called (r, n)-*path*). Suppose that indeed $n \in T(E_{i_1})$, then P is said to be

a *directed cycle* (or *circuit*); this is in particular true when $r = n$. The length of P is equal to the number of hyperarcs on it, denoted by $l(P)$. Moreover, we'll denote $V(P)$ and $E(P)$ by the set of vertices and hyperarcs respectively traversed via P. For example, in the directed hypergraph of Fig. 2.9, we have $Q = (x_2, E_1, x_3, E_2, x_7, E_5, x_8)$ is a directed path from x_2 to x_8 of length 3, where $V(Q) = \{x_2, x_3, x_7, x_8\}$ and $E(Q) = \{E_1, E_2, E_5\}$. If a directed path exists from r to n in a directed hypergraph H (i.e an (r, n)-path exists), we say that n is connected to r in H.

B-connection and B-path [70] The concept of *B-connection* in directed hypergraphs is captured by the following definition.

Definition 2. B-connection to v in H.

1. Vertex v is B-connected to itself;
2. If for some $E \in E(H)$ all the vertices in $T(E)$ are B-connected to v, then each vertex $u \in H(E)$ is B-connected to v.

A *B-hyperpath*, or simply *B-path*, from node v to node x in a directed hypergraph H is a minimal (with respect to inclusion) subdirected-hypergraph of H where x is *B-connected* to v according to definition 2. A B-path can be defined as a sequence of hyperarcs used to prove that x is B-connected to v. The concept of a B-path generalizes the notion of a directed path in digraphs.
As an example, a B-path which B-connects vertex x_8 to vertex x_3 in the directed hypergraph of Fig. 2.9 is the subdirected-hypergraph H' defined by the vertex set $V(H') = \{x_1, x_3, x_7, x_8\}$ and the set of hyperarcs $E(H') = \{E_2, E_4, E_5\}$.

Cuts and Cutsets Let $s, t \in V(H)$. A *cut* T_{st} of H is a partition of $V(H)$ into two subsets X and Y such that $s \in X$ and $t \in Y$. Given the cut T_{st}, its *cutset* E_{st} is the set of all hyperarcs satisfying that $T(E) \subseteq X$ and $H(E) \subseteq Y$. The *cardinality of a cut* is the cardinality of its cutset. The same definition of cuts and cutsets applies to digraphs.

weighted directed hypergraphs A *weighted directed hypergraph* H is termed *weighted* if there is a real weight vector $w(E)$ assigned to each hyperarc E of H (via a real vector function w). Such a directed hypergraph is denoted by (H, w). Usually, scalar weights are considered for the sake of simplicity.

2.5 The theory of complexity

Any posed problem possesses a certain complexity imposed when trying to solve it. In this section, we introduce different notions to evaluate and estimate the complexity of a problem, discover the level of its difficulty, and construct the most adequate solution methods compatible with the relative complexity of the problem.

A problem is characterized by a general representation of the parameters and variables whose values are left unspecified, and a statement of what properties its solution is required to satisfy. An instance of a problem is obtained by specifying particular values for all the problem parameters. As an example, we'll introduce

the *Traveling Salesman Problem* (TSP) whose parameters consist of: a finite set $C = \{c_1, c_2, \cdots, c_m\}$ of cities, the distance $d(c_i, c_j)$ between each pair of cities c_i, c_j in C, and a certain starting point (say city c_1). A solution of TSP is an ordering $< c_1, c_{\Pi(2)}, \cdots, c_{\Pi(m)} >$ of the given cities that minimizes the weight of the cycle or tour (passing by all the cities once) imposed by this ordering, i.e which minimizes $d(c_1, c_{\Pi(2)}) + \sum_{i=1}^{m-2} d(c_{\Pi(i)}, c_{\Pi(i+1)}) + d(c_{\Pi(m)}, c_1)$. One instance of TSP is given by $C = \{c_1, c_2, c_3, c_4\}$, $d(c_1, c_2) = 3$, $d(c_1, c_3) = 1$, $d(c_1, c_4) = 6$, $d(c_2, c_3) = 4$, $d(c_2, c_4) = 2$, $d(c_3, c_4) = 5$ and the starting point c_1. The ordering $< c_1, c_3, c_4, c_2 >$ is the solution of this instance (min tour of length 11).

A decision problem Π is a problem which has only two possible solutions, either the answer "yes" or the answer "no". Abstractly, a decision problem Π consists simply of a set D_Π of instances and a subset $Y_\Pi \subseteq D_\Pi$ of yes-instances.
Many problems that arise in practice, such as TSP, are optimization problems rather than decision problems. Nonetheless, each problem implicitly includes an infinitude of decision problems. In fact, any optimization problem, whether a minimization or a maximization one, can be transformed into a decision problem by the use of a bound $B \geq 0$, then asking if we can find a solution which satisfies the problem's statement but whose cost doesn't exceed B in case the problem was a minimization one (or if the cost is at least as large as B in case we were dealing with a maximization problem). The optimization problem can be solved by repeatedly solving its decision problem version, by incrementally decreasing (resp. increasing) the value of B if the optimization problem version is a minimization problem (resp. maximization problem). The benefit of using decision problems will be highlighted later after having introduced some additional topics.

Accordingly, the decision problem version of TSP, for instance, can be stated as follows:

INSTANCE A finite set $C = \{c_1, c_2, \cdots, c_m\}$ of cities, a distance $d(c_i, c_j) \in Z^+$ for each pair of cities $c_i, c_j \in C$, a starting point, and a bound $B \in Z^+$.

STATEMENT Is there a tour passing by all the cities in C exactly once and returning back to the starting point, and whose total length is no more than B?

An algorithm, which consists of step-by-step procedures for solving problems and is written in some precise computer language, is said to solve a problem Π if it can be applied to any instance I of Π and is always guaranteed to produce a solution for that instance. For example, an algorithm doesn't solve TSP unless it always constructs an ordering that gives a minimum length tour.
We shall start by a formal definition of the concept of an algorithm based on what is known as a *Turing machine*, which is used as a computer model of computation because of its simplicity.

2.5.1 Deterministic Turing Machine and the class P

First, we'll begin by what's called a *Deterministic Turing Machine (DTM)*. It consists of an *infinite tape* that is divided lengthwise into cells on which the input will be written, and a *read-write head*. Note that the description of a problem instance that is provided as input to the computer is a single finite string of symbols chosen from a finite input alphabet, denoted by Σ. An additional symbol is required to form the symbols written on the *DTM tape* which is a distinguished blank symbol b. A program for a DTM specifies the following information:

- The finite set $\Gamma = \Sigma \cup \{b\}$ of tape alphabet.
- A finite set of states Q, with a specification of the initial state q_0 as well as the halting state(s) (forming set Q_h).
- A transition function $\delta : (Q - Q_h) \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$.

Each cell of the tape accepts exactly one symbol of Γ. Input to the machine comprises a tape which is blank except for a finite number of neighboring filled cells. The program starts its operation in state q_0 with the read-write head scanning the first non-blank symbol from Σ of the input's string. The computation proceeds then in a step-by-step manner. If the current state belongs to Q_h (i.e is a halting state), then the computation has terminated. Otherwise, the current state q belongs to $Q - Q_h$, some symbol $s \in \Gamma$ in a cell of the tape is read by the read-write head, and the value $\delta(q, s) = (q', s', \Delta)$. The tape's read-write head then erases s and writes s' in its place, changes the state of computation from q to q', and moves one square to the left if $\Delta = -1$ or one square to the right if $\Delta = +1$, thus completing one step. In this way, the computation of DTM is carried out in a cyclic manner. So, from any pair of (i) a present state and (ii) a symbol just read, DTM can determine exactly one new state, one symbol to be written, and one direction of movement. It's in this sense that the Turing machine is said to be *deterministic*.

The output of the calculations is provided by either the symbols printed on the tape when entering a final state (excluding the blank symbols from both sides of the tape) or the final state itself, or both. Normally for decision problems, a typical final state is either *yes* or *no*, represented by the states q_Y and q_N respectively (i.e $Q_h = \{q_Y, q_N\}$). For further details and examples, see [63].
We point out that a DTM program can be used to implement functions. Note that the set of all finite strings of symbols from Σ (resp. Γ) is denoted by Σ^* (resp. Γ^*). Suppose M is a DTM program with input alphabet Σ and tape alphabet Γ that halts for all input strings from Σ^*. Then M is said to compute the function $f : \Sigma^* \to \Gamma^*$ if for each $x \in \Sigma^*$, $f(x)$ is defined to be the string output of M.

Now we'll introduce the notion of efficiency of an algorithm, called computational complexity (complexity for short). In fact, different algorithms possess different complexities and the characterization of which of these are "efficient enough" and which are too "inefficient" conquers a wide importance. Usually, complexity expresses

the time requirements resource as it often represents a dominant factor determining whether a particular algorithm is efficient or not.

Informally, the complexity of an algorithm is a function C which represents the maximum number of basic computational steps (such as arithmetical operations and comparisons) required for its execution, taken over all possible problem instances of a given problem size s (C is a function of s). This is known as the *worst-case complexity*. In graph theoretic algorithms and away from the Turing machine's concept, problem size can be for example a function of the number of vertices in the graph or the number of edges or both. When considering the turing machine's computation, the size of a problem instance is defined by the number of symbols in the string representing it, which reflects the amount of input data needed to describe the instance. The size of an instance is also called the input length.

We say that a function f has order no greater than that of g, and write $f = O(g)$, if $\exists$ two constants k and n_0 such that $f(n) \leq kg(n)$, $\forall\ n \geq n_0$. Informally, an algorithm is said to be *polynomial* (resp. exponential) if its complexity function C is $O(f)$ for some polynomial (resp. *exponential*) function f. A polynomial-time algorithm is further qualified as *linear-time* if the polynomial is a linear function, *quadratic-time* if it is a quadratic function, etc $\cdots$. Polynomial algorithms are considered to be efficient algorithms, unlike the problems for which it is conjectured that no polynomial algorithm exists (called intractable problems). This is because as problem size increases, the number of steps required by a polynomial-order algorithm (as opposed to exponential-order ones) grows relatively slowly. Indeed, algorithms whose complexity is exponential in the size of the input have running times which render them unusable even on inputs of moderate size and even by using the fastest computers available. By definition, the class P is the set of all problems which can be solved by a polynomial-time algorithm.

A formal definition of the complexity of an algorithm and the class P can be provided using the turing machine's representation. The time used in the computation of a DTM program M on an input is the number of steps occurring in that computation up until a halt state is entered. For a DTM program M that halts for all input strings, the complexity function C is the maximum time (i.e number of steps) required when applied to all instances of a given input size s (i.e $C(s) = max\{m \backslash$ there is an input string of size s which requires time $m\}$). A DTM program is called a *polynomial time DTM program* if $\exists$ a polynomial p such that the complexity function C is $O(p)$. Finally, the class P is a set which includes all problems whose algorithms are realized by a polynomial time DTM program.

Another type of a turing machine was also defined, called the Non-Deterministic Turing Machine (NDTM). It's not an existing computing machine, but it was mostly used to describe intractable problems. All the aspects of this machine and the notion of it "solving" an intractable problem will be presented right in the following subsection.

2.5.2 Nondeterministic Turing Machine

In this part, we'll confine ourselves to the decision problem version of optimization problems which is useful especially when dealing with intractable problems. This is because the decision problem is no harder than the corresponding optimization problem. For example, if one can find a minimum tour for TSP in polynomial time, then the associated decision problem can be solved in polynomial time. All is needed is to find the minimum length tour, compute its length, and finally compare it with the given bound B. Consequently, if the decision problem was intractable, as is actually the case of TSP, then so will be its at least as hard associated optimization problem. Thus, even though the theory of intractability restricts attention to only decision problems, one can extend the implications of the theory to optimization problems.
A new class of problems, called Nondeterministic Polynomial-time (NP) problems, will be defined right next from both the formal and informal aspects.

2.5.2.1 The class NP

We start by a simple example to refer to the class NP. Consider again the previously presented TSP decision problem version. There is no far a reported polynomial time algorithm for solving this problem. However, consider a problem instance in which a particular ordering of the cities was produced for which it was claimed that it represented a hamiltonian cycle whose weight was less than the given bound B (i.e the answer to the decision problem for this particular instance is *yes*). It's straightforward to devise a polynomial verification algorithm to check whether or not the provided ordering is actually a spanning tour and, if so, compute its length and compare that quantity to the given bound B. It is in this sense when the problem is said to be *polynomial-time verifiable*. Notice that polynomial time verifiability does not imply polynomial time solvability. In saying that we can verify a "yes" answer for TSP instance in polynomial time, one is not counting the time one might have to spend in searching among the exponentially many possible tours.

Before considering the formal turing machine's computation process to represent this new class of problems, we'll provide an informal view.
A *nondeterministic algorithm* is an algorithm which consists of two stages, namely the *guessing* and the *checking* stage. Given a problem instance I, the first stage "guesses" some structure S, then both I and S are provided as inputs to the checking stage which proceeds to compute in a normal deterministic manner, either halting with answer "yes", or with answer "no", or computing forever without halting. Such an algorithm is said to be *polynomial* if for all yes-instances I of size s in Y_{Π}, there is a guess that causes the verification stage to terminate in a number of basic computational steps which is a polynomial function of s. For example, a *polynomial nondeterministic algorithm* for TSP could be constructed using a guessing stage that simply guesses an arbitrary ordering of the given cities and a checking stage that is identical to the aforementioned polynomial time verification for TSP.
Finally, the class NP is defined to be the set of decision problems which can be

solved by a *polynomial nondeterministic algorithm.* As a summary, one can prove that a decision problem is an NP-problem if for a given "yes" instance I and after guessing a certain solution S, it can be verified that the answer for I and S is "yes" in polynomial time.
We'll introduce a new type of a turing machine called the Nondeterministic Turing Machine (NDTM). We will present a slightly non-standard NDTM model. More standard versions are described in [99, 100]. It has the same structure as a DTM (including an infinite tape and read-write head), except that it is in addition augmented with what is referred to the *guessing module* having its own *write-only head.* An NDTM program is specified in exactly the same way as a DTM program, including the tape alphabet Γ, input alphabet Σ, state set Q indicating the initial state q_0 and the halting states q_Y and q_N, and a transition function δ. A problem instance is assumed written in the cells in the same input convention recorded for DTM. The computation of an NDTM program differs from that of a DTM in that it takes place in two distinct stages. The first stage, known as the *guessing stage*, constructs a guess (sometimes called *succinct*) using the *write-only head* and records it in the cells according to the same input convention. This construction is nondeterministic in the sense that it's completely chosen in an arbitrary manner. The computation in the second stage, called the *checking stage*, proceeds solely under the direction of the NDTM program according to exactly the same rules as for a DTM and verifies deterministically whether the answer is *yes* or *no* for both the guess and the input problem instance. Remark that in the checking stage (i) the guessing module and its write-only head are no longer involved, having fulfilled their role of constructing the guess (ii) the guess is examined. The program eventually halts at either state q_Y or q_N, or need not halt at all. Thus in an NDTM, instead of having just one possible computation on a given input, it has many different ones, one for each possible guess.

For an NDTM program, the time required to compute a yes-instance I of a decision problem Π is defined to be the minimum number of steps (over all computations halting at q_Y for I, called accepting computations) occurring in the guessing and checking stages up until the halt state q_Y is entered. The complexity function $C : Z_+ \rightarrow Z_+$ for an NDTM program M is defined such that $C(s) = max\{\{1\} \cup \{m \backslash$ there is an input string of size s which requires time $m\}\}$. This time complexity function depends only on the number of steps occurring in accepting computations and by convention, $C(s)$ is set equal to 1 whenever no inputs of size s halt at the state q_Y. We say that an NDTM program M runs in polynomial time if there exists a polynomial p such that $C = O(p)$. Finally, the class NP is formally defined as the set of decision problems whose algorithms can be realized by a *polynomial time NDTM program.*

In an analogous way, a decision problem Π is said to belong to the class co-NP (informally) if: For any instance I of Π whose answer is "no" (i.e $I \in D_\Pi \backslash Y_\Pi$), there is a succinct (or guess) which confirms that this is so, and can be verified in "polynomial" time. Following is an example of a problem which is both NP and co-NP, but note that decision problems need not always be both.

Consider the problem of determining whether a graph is bipartite, call it Π_1. Clearly, an instance of such a problem is a graph G, and the yes-no question which needs to be solved is to determine if G is bipartite or not. We have:

1. $\Pi_1 \in$ NP. In fact, a bipartition of G constitutes the succinct in this case: Given a bipartition (X, Y) of a bipartite graph G (i.e the answer is "yes" for G), it is easy to check (in polynomial time) if each edge of G has one end in X and one end in Y. If so, then we can answer yes; otherwise, a new guess is constructed which is also checked in turn.
2. $\Pi_1 \in$ co-NP. In this case, the succinct or the guess is an odd cycle: since every non-bipartite graph contains an odd cycle, then given a cycle of a non-bipartite graph G (i.e an instance whose answer is "no"), one can easily check its length in polynomial time. If the length is odd, then we have the "no" answer. Otherwise, a new cycle in G is selected.

One can draw the following relationship between P, NP and co-NP written as: P $\subseteq$ NP $\cap$ co-NP. In fact, if Π is a decision problem in the class P and M is a polynomial time deterministic algorithm for Π, we can obtain a polynomial time nondeterministic algorithm for Π merely by using M as the checking stage and ignoring the succinct (guessing stage).

Note that if a decision problem Π belongs to NP (or co-NP), this means that no one was able so far to find a polynomial time algorithm for Π and that only nondeterministic polynomial algorithms were successfully constructed. Probably, a polynomial time algorithm could be found one day for such a problem, as there is no proof yet that it doesn't exist. From this viewpoint rises one of the most fundamental open questions in all of mathematics, the Cook-Edmond-Levin conjecture: P $\neq$ NP. If P differs from NP, then the distinction between P and NP-P is meaningful and important. In this context, we will provide right next a characterization for the hardest intractable problems in NP-P, called NP-complete problems. But first, we will introduce a necessary common approach of problem-solving which describes a polynomial transformation from a problem to another.

2.5.2.2 Polynomial transformation

A *polynomial transformation* from a decision problem Π_1 to a decision problem Π_2, denoted as $\Pi_1 \propto \Pi_2$, is a function $f : D_{\Pi_1} \to D_{\Pi_2}$ that satisfies the two conditions:

1. f is computable by a polynomial time algorithm.
2. f transforms an instance I of Π_1 to an instance $f(I)$ in Π_2 such that the answer corresponding to I with respect to Π_1 is *yes* if and only if the answer corresponding to $f(I)$ with respect to Π_2 is *yes* (i.e $I \in Y_{\Pi_1}$ if and only if $f(I) \in Y_{\Pi_2}$).

Here is an example to obtain a more concrete idea of what this definition means. Consider the Hamiltonian cycle (HC) problem stated as follows:

INSTANCE A graph $G = (V, E)$.
STATEMENT Does G contain a Hamiltonian cycle?

We shall show that HC $\propto$ TSP. The function f is defined quite simply. Let a graph $G = (V, E)$ of order m be an instance of HC. The corresponding instance of TSP is defined by f as follows:

- The vertices of G constitute the set of cities C.
- Any vertex is the starting point, say vertex v_1.
- For any two cities $v_i, v_j \in C$, set $d(v_i, v_j) = 0$ if $\{v_i, v_j\} \in E$, and 1 otherwise.
- The bound B is equal to 0.

In an informal sense and away from the turing machine's computation, it's easy to see that this transformation f can be computed by a polynomial time algorithm since for each of the $m(m-1)/2$ distances $d(v_i, v_j)$ that must be specified, it's necessary to examine G to see whether or not $\{v_i, v_j\}$ is an edge in E. Thus the first condition is satisfied.
For the second property, we need to prove that G contains a Hamiltonian cycle $\Leftrightarrow$ $f(G)$ admits a tour of all the cities which has length no more than $B = 0$. First, suppose that $v_1 v_2 \cdots v_m$ is a Hamiltonian cycle of G. Then $< v_1, v_2, \cdots, v_m >$ is also the desired ordering of $f(G)$ which has total length no more than 0, because each intercity distance traveled in the ordering corresponds to an edge of G and hence has weight 0. Conversely, suppose that $< v_1, v_2, \cdots, v_m >$ is an ordering of all the cities in $f(G)$ with total length no more than B. The fact that $B = 0$ implies that each pair of successively visited cities must be exactly distance 0 apart. By the definition of $f(G)$, it follows that $\{v_i, v_{i+1}\}$ $(1 \leq i < m)$, and $\{v_m, v_1\}$ are all edges of G and hence $v_1 v_2 \cdots v_m$ forms a Hamiltonian cycle of G.

The significance of polynomial transformation comes from the fact that if $\Pi_1 \propto \Pi_2$, then a polynomial time algorithm for solving Π_2 can be converted into a polynomial time algorithm for solving Π_1, stated as the following lemma:

Lemma 1. *if* $\Pi_1 \propto \Pi_2$, *then* $\Pi_2 \in P$ *implies* $\Pi_1 \in P$.

Proof. see [63, 61] □

Consequently and since HC $\propto$ TSP, we can deduce that if TSP can be solved in polynomial time, then so can be HC and if HC is intractable, then so is TSP. Thus $\Pi_1 \propto \Pi_2$ means that Π_2 is "at least as hard" as Π_1.
Note that the "polynomial transformability" relation "$\propto$" is reflexive. Indeed, it's especially useful because it's proved to be a transitive relation [63, 61]. This means that if $\Pi_1 \propto \Pi_2$ and $\Pi_2 \propto \Pi_3$, then $\Pi_1 \propto \Pi_3$.

Two decision problems Π_1 and Π_2 are said to be *polynomially equivalent* if $\Pi_1 \propto \Pi_2$ and $\Pi_2 \propto \Pi_1$. In fact, the class P forms an equivalence class under this order and can be viewed as consisting of the computationally "easiest" problems. On the contrary, the class of NP-complete problems, denoted by NPC, forms another such equivalence class, distinguished by the property that it contains the "hardest" decision problems in NP. In the next part, we will introduce the class of NP-complete problem in some more details.

2.5.2.3 NP-complete problems

Definition 3. A decision problem Π is said to be NP-complete if:

- $\Pi \in$ NP.
- $\forall\ \Pi' \in$ NP, $\Pi' \propto \Pi$.

Lemma 1 leads to specify the NP-complete problems as the "hardest problems in NP". If any NP-complete problem can be solved in polynomial time, then all problems in NP can be analogously solved. Indeed, if any problem in NP is intractable (i.e no polynomial time algorithm can ever be found for this problem), then so are all NP-complete problems. An NP-complete problem Π has therefore the property that if P $\neq$ NP, then $\Pi \in$ NP-P or alternatively $\Pi \in$ P if and only if P = NP.

These requirements we have just described are rather too demanding to prove that a problem Π is NP-complete. One must show that every problem in NP transforms to Π which doesn't seem at all to be a feasible way of proof. However, if one possesses just one problem that is known to be NP-complete, an immediate consequence of the just stated definition and the transitivity of $\propto$ can be deduced to prove that a problem Π is NP-complete, in the following manner:

1. $\Pi \in$ NP.
2. $\exists\ \Pi' \in$ NPC, such that $\Pi' \propto \Pi$.

Note that a decision problem is said to be NP-hard if it satisfies only the second property, i.e it differs from NP-complete problems by just the fact that it doesn't need to be an NP problem. Hence, all NP-complete problems are also NP-hard.

Thus, we still need some first NP-complete. This honor goes to the "Satisfiability" problem, for short SAT, which is defined as follows: Let P be a set of n Boolean variables. A *truth assignment* for P is a function $t : P \to \{T, F\}$, which assigns either the value *true* or *false* for each Boolean variable in P. A clause L is of the form: $p_1 \vee p_2 \vee \cdots \vee p_r \leftarrow p_{r+1} \wedge p_{r+2} \wedge \cdots \wedge p_q$ where $p_i \in P$ for all $i = 1, \cdots, q$. This clause is true if at least one of the variables $p_1, \cdots, p_r$ is true when all the variables $p_{r+1}, \cdots, p_q$ are true, and it's false otherwise (i.e $p_1, \cdots, p_r$ are all false, and $p_{r+1}, \cdots, p_q$ are all true). Clause L can be alternatively expressed in a *disjunctive form* as $p_1 \vee p_2 \vee \cdots \vee p_r \vee \neg p_{r+1} \vee \neg p_{r+2} \vee \cdots \vee \neg p_q$. A collection U of clauses over P is said to be *satisfiable* if and only if there exists a truth assignment for P which makes all the clauses in U true; otherwise, it is *unsatisfiable*. The decision problem version of satisfiability is:

INSTANCE A set P of n Boolean variables, and a set U of m clauses over P.

STATEMENT Is there a satisfying truth assignment for U?

Theorem 9. *SAT is NP-complete.*

Proof. It's easy to see that SAT $\in$ NP. A polynomial nondeterministic algorithm for it needs only guess a truth assignment for P and verify to see whether that assignment satisfies all the clauses in the collection C, which can be easily done in polynomial time. The rest of the proof involves the notion of a Turing machine (Cook 1971), which proof can be found in [101] or [63]. □

Starting from SAT and transforming one NP-complete problem to another, it has been demonstrated that the previously presented problem HC is NP-complete [61, 63]. Since HC $\propto$ TSP (see 2.5.2.2), then we deduce that TSP is also an NP-complete problem.

Many of the best algorithms (in terms of complexity) for solving intractable problem, the NP-complete problems for example, include a number of polynomial-order sub-algorithms. However, they are inefficient because these sub-algorithms must be used a number of times, where this number is of exponential order. An example of such an algorithm can be found in [61], chapter 9.

Hence, most intractable problems are usually solved by rather using approximate methods, called *heuristics*. A *heuristic* is a computational procedure based on the simple rule that one should usually yield a good approximate solution to the problem at hand rather than the best solution ever. One particularly simple and natural class of heuristics is the class of *greedy heuristics*. This is a simple-minded strategy of progressively building up a solution, one element at a time, by choosing the best current option at each iteration without regard to future consequences. This means that they make locally optimal choices, assuming optimal sub-solutions are always part of the globally optimal solution [72]. If this was the case, the procedure is called *greedy algorithm*. You will find examples of such an approach and others in the next section, in which we will exhibit several major graph theory applications. We will present various problems which were modeled either as graphs, digraphs, hypergraphs or directed hypergraphs. Indeed, we will provide the attained results on the complexity of each of these problems and accordingly, present their associated proposed algorithms and possible solutions (if any).

2.6 Graph theory applications

In this section, we will first start presenting some problems related to graphs and digraphs, and then move to others which were modeled as either hypergraphs or directed hypergraphs.

2.6.1 Graph and digraph problems

Traveling Salesman Problem (TSP) [71] This is the most famous "optimization problem". It's a problem closely related to the question of Hamiltonian cycles stated again as follows: a salesman wants to travel a set of cities during the trip. Given the distances between the cities, in what order should he travel

so as to visit every city precisely once and returns to the starting point, with the minimum total distance?
This problem can be formulated as a graph theoretic model by a complete weighted graph (K_n, w), devised with vertices, edges and weights representing the cities, the roads between them, and a real number weight (say the distance in miles) respectively. The question is to find a Hamiltonian cycle C in K_n s.t $w(C)$ is minimum.

It's clear that there are $(n-1)!$ cycles in K_n from a specific starting point. More precisely, we can only consider $(n-1)!/2$ cycles, since the same distance can be traveled in a reverse order.
The straight forward way to solve an instance of TSP is to examine all the $(n-1)!/2$ possible Hamiltonian cycles and then select the one which has minimum weight. However, finding such a cycle usually involves great labor when the number of vertices is large; for instance, for $n = 25$, a total of $24!/2$ (approximately 3.1×10^{23}) different Hamiltonian cycles would have to be considered and if it just takes one nanosecond (10^{-9} seconds) to examine each, a total of ten million years would be required to find the desired cycle by exhaustive search technique. Thus, several heuristic methods of solution were proposed instead that give a route very close to the shortest one, but do not guarantee the shortest (see [73, 82]). The simplest and most obvious construction heuristic is the *nearest neighbor*. The nearest neighbor is a greedy algorithm which builds a cycle by always choosing as the next vertex to visit, one that is nearest to the last one visited.

Depth-First Search (DFS) and Breadth-First Search (BFS) [62] These are the
two most important search methods which methodologically explore the edges of a given graph (resp. digraph) such that every edge (resp. arc) and each vertex is visited at least once. It's evident that almost any graph question requires examination of every edge (and in the process every vertex) at least once. For example, before declaring a graph G to be disconnected we must have visited every edge in G for otherwise, it might happen that the one ignored edge could have made the graph connected.
DFS and BFS trace each connected component of a graph or digraph G. Both search techniques run in polynomial time. They search respectively the edges of a graph as they move from one vertex to another as follows:

- BFS methodology: once at vertex v of G, examine all the edges (or arcs) incident with v and then pass to another vertex of G which is adjacent to v.
- DFS methodology: once at vertex v of G, scan a single edge (or arc) incident with v (which possibly still has unscanned edges) and then pass to the other vertex in G which is incident with this edge (or arc).

DFS is sometimes also called *backtracking*. This is because if it falls at a vertex v where at least one of its neighbors exists such that it is not yet examined by this technique, then it passes to one of these neighbors. If not, it *backtracks* to

the vertex which was just examined before v and again checks its neighbors, and so on. DFS is studied in some depth by Tarjan in [77].

The BFS approach is the archetype for many important graph algorithms, including Prim's minimum-spanning-tree and Dijkstra's single-source-shortest-path algorithms (introduced later in this section) etc $\cdots$. DFS appears also to be a useful approach, for example to check connectivity in graphs, isomorphism, planarity etc $\cdots$. See [61] for further information.

Minimum Spanning Tree (MST) A railway network connecting a number of cities is to be set up with the objective of making it possible to travel by some path between every pair of cities. Given the cost of construction w_{ij} of linking cities v_i and v_j, *the connector problem* involves designing the network with the minimum possible construction cost. The problem can be formulated as a graph theoretic model in which a weighted graph G is devised with vertices, edges, and weights representing cities, feasible connections, and construction costs respectively. The connector problem is equivalent to finding a minimum weight spanning tree of G [61], which problem is described right next.

The MST problem amounts to finding, in a weighted connected graph G, a connected spanning subgraph of minimum weight. If the weights are positive, this subgraph will be a spanning tree [62] (Note that a spanning tree T is a minimal connected spanning subgraph of a graph G, because every edge which remains in T is a cut edge). Thus, MST is stated as follows:
Given: a weighted connected graph G,
Find: a minimum-weight spanning tree T in G.

Several methods were proposed to solve the MST problem, both by hand and by computer. We will briefly present the two most famous polynomial time algorithms, namely Kruskal [74] and Prim [75] algorithms. Although they are both greedy algorithms and are thus not supposed to guarantee finding globally optimal solutions, but however, it has been proved that in fact they do yield a spanning tree with minimum weight [62, 71].
These two algorithms exploit two different ways to form a desired minimum spanning tree. In Kruskal's algorithm, all the edges of the graph G are listed in order of nondecreasing weight. At each successive step, it selects (from the remaining unselected edges of G) the smallest weight edge that makes no cycle with the previously selected edges, up until $n-1$ edges have been selected, which will constitute the desired minimum-weight spanning tree. As for Prim's algorithm, the tree starts from a certain specified vertex of the graph G to which is added, in a successive manner, a least-weight edge (not previously added) connecting the so far attained tree to a vertex not in the tree.

Minimal spanning tree has been found quite useful in providing a lower bound on the length of the traveling salesman's route in [76].

Shortest path problem A *shortest-path tree* T in a graph G (resp. digraph) is a tree (resp. out-branching), rooted at a vertex s (resp. with source s), that

contains all the vertices in G which are reachable from s [82]. Indeed, T is described such that the path in T from s to any vertex x in T is a shortest path in G, i.e $d_G(s,x) = d_T(s,x)$.
The BFS method is capable of finding shortest-path trees in graphs (it might also build a forest in case the graph was disconnected). Similarly, directed BFS is a straightforward analog of BFS for finding shortest-path trees in digraphs [62]. Both methods require polynomial time of execution.

Now we'll move to the weighted version of the shortest-path problem [71]. Let (G,w) be a weighted graph or digraph. There are different types of shortest-path problems. Most frequently encountered include (i.) shortest path between two specified vertices (ii.) shortest paths from a specific vertex to all others (iii.) shortest paths between all pairs of vertices, just to mention some. A good comparative study of various shortest-path algorithms through the year 1968 can be found in [79].
Among several algorithms that were proposed for the shortest path between a specified pair of vertices, the most efficient one is an algorithm due to Dijkstra introduced in [78, 71], whose computation time is proportional to $(|V(G)|)^2$.
An algorithm (by Dijkstra as well) solves the shortest path problem from a specific vertex to all others but assumes that all edge (resp. arc) weights in the input graph (resp. digraph) are nonnegative. Others, such as the Bellman-Ford algorithm, allow negative-weight edges (resp. arcs) and hence cycles (resp. circuits) of negative weight in the input graph (resp. digraph) [62]. Both algorithms run in polynomial time but Dijkstra's algorithm is faster than that of Bellman-ford.
Sometimes one is interested in finding the shortest paths between all $n(n-1)$ ordered pairs of vertices in a digraph (or $n(n-1)/2$ unordered pairs of vertices in a graph). The two polynomial algorithms considered to be the best for such a problem include one which is due to Dantzig [80] (which allows negative edge weights) and another due to Floyd [81, 71] (which doesn't allow cycles of negative weight). Both algorithms require computation time proportional to $(|V(G)|)^3$.
In almost all these cases, the presented algorithms generate rooted minimum-weight trees (or out-branchings) also can be called shortest-path trees.

Many real-world problems can be modeled as shortest path problems. For instance, one might wish to determine a shortest route between two specified locations in a city. This amounts to finding a path (resp. directed path) of minimum weight connecting two specified vertices in a weighted graph (resp. digraph), whose vertices are the road junctions and whose edges (resp. arcs) are the roads linking these junctions.

Graph coloring problem This problem is stated as follows: Given a graph $G = (V,E)$, find a function $f : V \to \{1,2,\cdots,k\}$ such that $f(u) \neq f(v)$ whenever $\{u,v\} \in E$ and such that k is as small as possible. This problem is proved to be an NP-hard problem (see for example [63]). However, several efficient variants of greedy heuristic coloring algorithms were proposed. We mention

the "one-pass" method, in which one runs through the vertices in order and always assigns the smallest possible color. As for the "many-passes" method, it runs through the vertices assigning color 1 whenever possible, then repeat with color 2 and so on. Both methods can verify that $\chi(G) \leq \Delta(G) + 1$ [83].

Channel assignment problem This is a telecommunication problem which can be reduced in some particular cases to the graph coloring problem. The radio channel assignment problem is the final stage in the design of a cellular radio communication system, described as follows: The service region is divided into cells around each transmitter (or base station). Channels are assigned to the different cells, in which the same radio channel can be used simultaneously in many cells, as long as they are sufficiently well separated to avoid interference. Since the radio spectrum is a finite resource which is heavily in demand, the question is to assign the channels to the transmitters carefully in order to take maximum advantage of this re-use possibility. Thus, the aim is to find an assignment of channels to the different transmitters such that the corresponding interference is acceptable, while using the minimum number of channels possible. Interference can be viewed as the difference between two channels; the smaller the difference, the greater the interference. For each pair u, v of distinct transmitters, there is a set T_{uv} of forbidden differences $|i - j|$ for channel i at u and channel j at v. The minimum distance for which interference is considered to be acceptable between transmitters u and v is denoted by m_{uv}.

This problem can be theoretically modeled as a weighted graph G. It assigns a vertex for each transmitter and distinct vertices u and v are adjacent if T_{uv} is a non-empty set; the weight assigned to each uv edge is m_{uv}. If the interference increases with the proximity between two channels, then $\forall$ $e \in E(G)$, T_e is of the form $\{0, 1, \cdots, m_e - 1\}$.
A channel assignment is a mapping $\phi : V(G) \to \{1, \cdots, t\}$. It's said to be a *feasible assignment* if $|\phi(u) - \phi(v)| \geq m_{uv}$ for every uv edge. The *span* of the problem, denoted by span*(G,m)*, is the least integer t such that there is a feasible channel assignment. For example, if G is the triangle K_3 with each edge of weight 3, then the span is 7. The assignment problem states to determine or approximate the span, and to find corresponding assignments.

It has been noted that when the weights assigned to all the edges e of G are the same, the problem is almost back to coloring [83]. In particular, if $m_{uv} = 1 \ \forall \quad e \in E(G)$, then span(G,m) (also denoted by span(G,1)) equals the chromatic number $\chi(G)$. Thus, this special case returns exactly to the graph coloring problem, which is NP-hard as indicated before. Consequently, we can not expect an easy ride to solve the assignment problem, since the general case is harder than graph coloring. A general exponential algorithm for the channel assignment problem was proposed in [84]. However, determining the span and finding an associated assignment is proved to be easy in some particular cases, i.e when the graph is bipartite or an odd cycle [83].

2.6.2 Hypergraph and directed hypergraph problems

An application of proper coloring for mixed hypergraphs [67] For cellular telephones, each zone is assigned a certain frequency selected from a specific list of frequencies associated to the zone. If two zones interfere, they can't use the same frequency at any time. Suppose we have n zones $z_1, z_2, \cdots, z_n$ and n lists of frequencies $L_1, L_2, \cdots L_n$ for the n zones respectively. First, begin by constructing a graph G with n vertices $z_1, z_2, \cdots, z_n$ and in which two vertices are adjacent if the respective zones interfere. Now, the problem is formulated as follows: In which ways can the vertices of G be colored such that adjacent vertices have different colors and each vertex is assigned a color from its list? To lose ambiguity, colors are named the same way as the existing frequencies. Such graph coloring is called the *list coloring*. Now in turn, this list coloring problem can be reformulated in terms of the proper coloring of a mixed hypergraph H, defined as follows: the set of vertices $V(H)$ include the n zones together with all the allowed frequencies in the n zones (i.e $V(H) = \{z_1, z_2, \cdots, z_n\} \cup L_1 \cup L_2 \cup \cdots \cup L_n$). The D-hyperedges of H include all edges of G, plus all edges (i.e two-element subset hyperedges) forming a complete graph on the vertices $L_1 \cup L_2 \cup \cdots \cup L_n$. As for the C-hyperedges, they include hyperedges of the form $\{z_i\} \cup L_i, \quad \forall\ i = 1, \cdots n$.

One can easily see that G admits at least one list coloring if and only if the associated mixed hypergraph H admits a proper coloring. In fact, since each D-hyperedge must have at least two vertices of different colors and since the D-hyperedges on $L_1 \cup L_2 \cup \cdots \cup L_n$ form a complete graph, then this means that no two vertices in this complete graph will be assigned the same color. Moreover, since every C-hyperedge must have at least two vertices of the same color, then each z_i vertex will be obligatory assigned one color (i.e one frequency) already assigned to one vertex in the list L_i. Finally, the edges of G which are D-hyperedges in H guarantee that no two interfering zones have the same color.
A greedy algorithm has been presented concerning the colorability of mixed hypergraphs and some work was done on special cases of colorable and uncolorable ones in [66].

Note that this problem is a particular case of the lately described channel assignment problem, in which a certain list of channels (or frequencies) is assigned to each cell (or zone) representing the channels that are allowed to be used in the cell, and where the interference between two cells, when it exists, only occurs if they are assigned the same channel (i.e at distance zero).

Directed hypergraph problems and some of their applications First, some problems related to visiting a directed hypergraph starting from an origin node r were worked out in [69]. By visiting it's meant finding all the nodes which are connected (resp. B-connected) to r. The algorithm *Visit* (resp. *B-Visit*) finds all such nodes and returns a set of paths connecting (resp. B-connecting) them to r. These paths developped by this algorithm define what's called a *tree*

(resp. *B-tree*) rooted at r. Both *Visit* and *B-Visit* run in linear time.
Next, we will highlight the problem of finding minimum weight B-paths in a weighted directed hypergraph. For this, we will need to introduce some new aspects.

Let (H, w) be a weighted directed hypergraph and let $\Pi = (V_\Pi, E_\Pi)$ be a B-path B-connecting vertex t to vertex s in $V(H)$. A *weighting function* of Π is a node function W_Π which assigns weights to all its nodes depending on the weights of its hyperarcs. $W_\Pi(t)$ represents the weight of the B-path Π under the chosen weighting function. Usually, the weighting functions are defined such that $W_\Pi(s) = 0$ and $W_\Pi(y)$ depends only on the hyperarcs preceding y in the B-path Π, $\forall \quad y \neq s$. A typical example of this kind of weighting functions is the *Cost*, C_Π, defined as the sum of the weights of all the hyperarcs preceding node y in Π, i.e $C_\Pi(s) = 0$ and $C_\Pi(y) = \sum_{E \in E_{\Pi_{sy}}} w(E)$ for $y \in V_\Pi \backslash \{s\}$, where Π_{sy} is the B-path in Π B-connecting y to s (note that $C_\Pi(t) = \sum_{E \in E_\Pi} w(E)$ is the cost of Π). A special class of such weighting functions, called *additive weighting functions*, is defined such that the weight of node y can be written as a function of both the weights of the hyperarcs entering into y and that of the nodes in their tails, i.e:

- $W_\Pi(y) = min\{w(E) + F(\{W_\Pi(x) : x \in T(E)\}) : E \in E_\Pi \cap BS(y)\}$, $y \in V_\Pi \backslash \{s\}$,

where F is a non-decreasing function of $W_\Pi(x)$ for each $x \in T(E)$.

Unfortunately, at least in general, the problem of finding a minimum weight B-path in a weighted directed hypergraph is NP-hard because Italiano and Nanni [85] showed that the particular problem of finding minimum cardinality B-paths in a B-graph is NP-hard. Nevertheless, many particular cases exist for which the problem is easy to solve. One example is when the weighting functions are additive, for instance see [86]. We will exhibit right next one example of such a case, presented in [69].

An algorithm, called Shortest B-Tree *SBT*, solves the problem of finding a set of minimum weight B-paths from origin r to all the nodes y which are B-connected to r, which is related to the problem of finding a solution to the *Generalized Bellman's Equations* (defining an additive weighting function) written as:

$$\begin{aligned} &W(r) = 0 \\ &W(y) = min\{w(E) + F(\{W(x) : x \in T(E)\}) : E \in BS(y)\} \quad \text{y} \neq r \end{aligned} \tag{2.1}$$

This is in fact a generalization of the well known shortest path tree problem in weighted digraphs. Procedure SBT finds a solution of 2.1 and returns what is called a minimum weight *B-tree* rooted at r. If y is not B-connected to r, SBT returns $W(y) = +\infty$. SBT runs in polynomial time in all of its 3 variants discussed in [69].

Several directed hypergraph applications naturally arise. We will briefly mention some of them:

Satisfiability To a given instance $\Pi \in SAT$ one can associate the directed hypergraph H_Π, with one node for each element in the set P of Boolean variables and one hyperarc E with $H(E) = \{p_1, p_2, \cdots, p_r\}$ and $T(E) = \{p_{r+1}, p_{r+2}, \cdots, p_q\}$ for each clause $p_1 \vee p_2 \vee \cdots \vee p_r \leftarrow p_{r+1} \wedge p_{r+2} \wedge \cdots \wedge p_q$. A theorem proved in [69] gives a characterization of a satisfiable instance Π of SAT related to cuts of the associated directed hypergraph H_Π. Recall that SAT is an NP-complete problem.

A particularly important case is when a clause is such that $r \leq 1$, i.e the clause is of the form $p_1 \leftarrow p_2 \wedge p_3 \wedge \cdots \wedge p_q$. Such clause is called a *Horn clause*. HORN-SAT is a particular case of SAT problem, in which all the clauses of its instances are Horn clauses. Clearly from the definition, if $\Pi \in HORN - SAT$ then H_Π is a B-graph. This particular satisfiability *HORN-SAT* problem is polynomial since it can be solved in linear time [91, 92]. Indeed, HORN-SAT is shown to be equivalent to the problem of finding a B-path in a B-graph in [69]. Then, B-Visit can solve any instance of *HORN-SAT* in linear time which was another way to prove that *HORN-SAT* can be easily solved.

Relational data bases A substantial amount of research has been devoted to the analysis of relational data bases some of which directed hypergraphs provided a natural and unifying formalism [69, 94, 95, 96].

Urban transit application The analysis of passenger distribution in an urban transit system was shown to be an interesting application of F-graphs in [97, 98].

2.7 Conclusions

In this chapter, we have introduced two major theories: graph theory and the theory of complexity. Both theories render importance to our work; the first in order to supply a theoretical graphical modeling to the problem of designing an optimal SDR multi-standard system, and the second to explore and estimate the complexity of the problem we're dealing with. This will form a new application of graph theory to our stated telecommunication problem, where our final objective is to find new appropriate theoretical tools capable of solving it. All these topics will be examined and detailed in the subsequent chapters.

Chapter 3

A theoretical study of the problem related to SDR multi-standard systems

Contents

In chapter one, we have introduced a graphical approach of the SDR multi-standard system which provides a pictorial view of all the options capable of implementing the multi-standard design, as well as presented a cost function equation which calculates the cost of any selected alternative of implementation. However, in this chapter, we will model all these aspects and others theoretically using graph theory or more precisely, using directed hypergraphs. This will constitute the first section of this chapter.

Our aim is to help extract the most suitable COs inside and between the standards, which guarantee the best trade-off between flexibility and efficiency. This can be done by optimizing the proposed cost function in order to find the option that has

a minimum cost. This optimization problem was tackled previously by other PhD students but its complexity was neither examined nor studied. Thus in the second section of the present chapter, we perform an exploration of the total number of options that are capable of implementing the multi-standard system, which will represent an inspiration for the difficulty of the problem we're dealing with. Afterwards in the third section, we study the complexity of our optimization problem. In fact, we prove that the decision problem version of this optimization problem is an NP-problem under a certain specified constraint. Finally, a conclusion's section ends this chapter.

3.1 A formal model for different aspects of the SDR multi-standard terminal

In this section, we will present various theoretical aspects related to the SDR multi-standard design. More precisely, we first model the graph structure of the SDR multi-standard system (explained in chapter one) as a directed hypergraph, and then present a suggestion for a graphical representation of any selected option of implementation. Afterwards, we provide an explanation of the theoretical multi-standard graphical illustration which actually results from the theoretical graphical break-down of each of the supported standards separately. Finally, we derive an alternative formal theoretical expression of the proposed cost function (presented in chapter one) as function of various definitions and notations concerning directed hypergraphs.

3.1.1 A mathematical model of the graph structure of the SDR multi-standard system

Fig. 3.1 (previously introduced in chapter one), which represents the graph structure of the multi-standard system supporting S & T, is reconsidered here on which we additionally associate numerical values to some nodes which represent the BC/CC of the node, and others associated with the BF-reductions standing for the NoCs. All the blocks and arcs in the figure are associated with these numerical parameter values but only some of them are indicated in Fig. 3.1 for simplicity reasons. Note that these numerical values are arbitrarily chosen and follow a certain logic on relationships between BCs and CCs of higher to lower hierarchy blocks (as explained in chapter 1).

The graph structure of the theoretical approach of parametrization presented in chapter 1 can be given a formal approach. Formally speaking, the graph structure of a multi-standard system can be theoretically modeled as a **directed hypergraph** H defined by the couple (V, E), where the set of vertices V includes the blocks (functions and operators) present in the graph structure (example $V = \{S, T, A1, A2, A3, B1, \cdots, D4, D5\}$ in the graph structure of Fig. 3.1). As for the set of hyperarcs E, it is formed such that a directed hyperedge $e \in E$ includes a parent node as the only tail node while all the necessary descendent node(s) capable

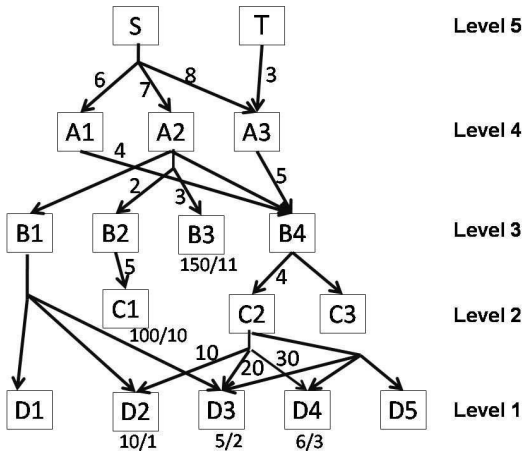

Figure 3.1: Global structure of a multi-standard system showing the break-down of standards S and T up to 4 lower levels

of performing this parent's node task constitute the head node(s) of e. This means that whenever we have an "AND" dependency, the corresponding hyperarc is formed such that the parent node constitutes the tail node while all the descendent nodes via this "AND" dependency form its head nodes. Whereas when faced with an "OR" dependency, the hyperarc will as well have the parent node as the only tail node, while only one of its descendent nodes (if more than one exists) via the corresponding "OR" dependency will constitute the hyperarc's head node. In this way, we form the set of hyperarcs E including all the "OR" and "AND" dependencies present in the corresponding graph structure. For instance, we have $(\{S\}, \{A1, A2, A3\})$, $(\{B4\}, \{C2\})$, $(\{B4\}, \{C3\})$, $(\{A2\}, \{B1\})$, $(\{A2\}, \{B2, B3\})$, $\cdots$ etc belong to the set of hyperarcs E of the directed hypergraph of Fig. 3.1.
Remark that this directed hypergraph representation of the multi-standard system is in fact an F-graph since each hyperarc contains only one tail node which is the parent node.

Some numerical values are indeed associated with different entities of such a directed hypergraph representation $H = (V, E)$. There are the BC and CC parameters associated with each vertex $x \in V$, and NoC associated with each BF-reduction of any hyperarc $e \in E$, called an arc of e. We'll refer to these positive values as weights associated with the nodes and the arcs of H. However, note that H doesn't correspond to a weighted directed hypergraph because the weights are associated with the arcs instead of the hyperarcs of the directed hypergraph, and indeed weights are associated with the nodes (check the definition of weighted directed hypergraphs in chapter 2).

Each PE included in the graph structure of a multi-standard system (represented as an F-graph H) occupies a certain absolute level. The following is a small remark on the level assignment of each PE. Suppose that we're decomposing the supported

standards up to $n-1$ different lower levels (thus as a whole, we'll have a total of n levels including the top level standards). The level of a block v in the directed hypergraph representation of a multi-standard system, denoted by $L(v)$, is defined by:

$$L(v) = n - \max_{x/d_H^-(x)=0} (\max_{P\,:\,xC-path}(l(P))), \tag{3.1}$$

where $l(P)$ stands for the length of the path P.

As an example, we will identify the level of the "$C1$" block in Fig. 3.1, where we have $n = 5$ levels in this figure (because the decomposition is up to $4 = n-1$ lower levels break-down of the supported standards S and T). The level of the "$C1$" block is equal to 2 because:
We choose a block x whose in-degree is zero (which stands for one of the top level standard blocks), say we select S. Then find the maximum length among all the paths from S to $C1$. In this case we only have one (passing through $A2$ & $B2$), whose length is 3. Select another block x of in-degree zero (the remaining block to choose is T) and again find the maximum length among all paths from T to $C1$, in which case we have no such path. Finally the maximum length of all these paths will be 3, and so $L(C1) = 5 - 3 = 2$ according to the definition of equation 3.1.
The level of each block in Fig. 3.1 is identified on the right side of the figure.

For all what follows, we'll write an xy-path to mean (x, y)-path which is a directed path from x to y in a directed hypergraph H. Moreover, we'll denote $\Im$ by the set of the top level standards in the multi-standard system. So, $\Im = \{S, T\}$ in the case of Fig. 3.1.

The graph structure of the multi-standard system supplies us with all the options that can implement the design. Next, we are going to explain how any one of these options of implementation can be graphically illustrated. But before, we will introduce several necessary definitions for the rest of our work concerning directed hypergraphs.

Definition 4. Weight of a path
Let $P = P_{rn} = (v_1 = r,\, e_{i_1},\, v_2,\, e_{i_2},\, v_3,\, \cdots,\, e_{i_q},\, v_{q+1} = n)$ be an rn-path and let $e_{i_j} \in E(P)$. We'll define the BF-reduction of e_{i_j} via the path P by its particular BF-reduction obtained by selecting the predecessor vertex to e_{i_j} in the path P as its specific tail node and the successor vertex to e_{i_j} in P as its head node. Denote $BF_P(e_{i_j})$ by this BF-reduction of e_{i_j} via P. Then according to the definition, we get: $BF_P(e_{i_j}) = (\{v_j\}, \{v_{j+1}\})$, $j = 1, 2, \cdots, q$.
Suppose that we have a directed hypergraph $H = (V(H), E(H))$ in which a positive integer weight is assigned to every BF-reduction of any hyperarc in $E(H)$. For every P a path from r to n, we'll denote the weight of P by the product of the weights of the BF-reductions via P of all the hyperarcs in $E(P)$. So we can write:

$$w(P) = \prod_{e_{i_j} \in E(P)} w(BF_P(e_{i_j})), \tag{3.2}$$

where $w(P)$ denotes the weight of the path P and $w(BF_P(e_{i_j}))$ stands for the weight of $BF_P(e_{i_j})$ in H.

For example, the weight of the only path from S to $C1$ in Fig. 3.1 (passing through $A2$ and $B2$) is $w(\{S\},\{A2\}) \times w(\{A2\},\{B2\}) \times w(\{B2\},\{C1\}) = 7 \times 2 \times 5 = 70$.

Definition 5. Hyperarc addition
Let X be a subdirected-hypergraph of a directed hypergraph H s.t $E(X)$ different than $E(H)$, and let e be a hyperarc in $E(H)$ but not in $E(X)$. By adding e to X we obtain a subdirected-hypergraph $X^{'}$ of H, denoted by $X + e$, defined such that: $V(X^{'}) = V(X) \cup H(e) \cup T(e)$ and $E(X^{'}) = E(X) \cup \{e\}$.
$X + e$ is called a subdirected-hypergraph of H induced by X and e.

Definition 6. G-path
Let H be a directed hypergraph and $N \subseteq V(H)$.
We say that a subdirected-hypergraph X is a G-path of H with root N if it satisfies:

1. $d_X^+(u) \in \{0,1\} \quad \forall\, u \in V(X)$
2. $N \subseteq V(X)$
3. $\forall\, u \in V(X)$, $\exists$ a path from v to u for some $v \in N$

3.1.2 A representation of one option of implementation

Recall that a certain selected option is characterized by the chosen common operators to install inside the design. A pictorial view of a specific realized option of implementation will be defined by a **generated graph (GNG)**, which is a graphical illustration of one alternative capable of implementing the SDR multi-standard system. It is obtained from the directed hypergraph representation of the graph structure of a multi-standard system by plotting the node blocks chosen to install in the design such that they have empty forward stars, along with all the operators that they build, step by step, until we reach the functionalities of the top level standards. Consequently, the out-degree of the common operators X which are chosen to be installed in the terminal will be zero in the associated generated graph, while the out-degree of the remaining operators that they build will be exactly one.

Fig. 3.2 shows the generated graphs (obtained from Fig. 3.1) of two different options of implementation capable of realizing S and T. In the first option, the chosen operators are $D2, D3, D4, C1, \&B3$ and the corresponding GNG is illustrated on the left side of Fig. 3.2. As for the second option, the chosen operators to install inside the design are $D2, D3, D4, C1, B3\&B4$ and the GNG representation of this option

is the one pictured on the right side of Fig. 3.2. We'll explain how the first choice is capable of implementing S and T technically. Blocks $C1$ and $B3$ are installed to establish the functionality of block $A2$. Actually, we can get the $B2$ functionality from the $C1$ block by calling it a certain number of times (5 times in our case), and then calling this attained functionality of $B2$ many times (2 times), together with block $B3$ (called 3 times), can perform some specific tasks similar to the ones executed by block $A2$. In an analogous way, we see that blocks $D2, D3$ and $D4$ can perform the tasks of $A1$ and $A3$ via the $C2$ and then the $B4$ functionalities, where we have to multiply the computational costs of $D2, D3$, and $D4$ by the corresponding number of calls. Finally, having realized the functionalities of $A1, A2$, and $A3$, we can achieve the S block's functionality by calling them a NoCs times (6, 7, & 8 times respectively). However, only the $A3$ functionality is required to realize the functionalities of T. In a similar manner, one can deduce from the technical point of view how the second choice is intended to implement the multi-standard system supporting S and T.

The only difference between these two choices is that in the first, blocks $D2, D3, \& D4$ are used to implement the functionalities of both $A1 \& A3$ passing through the $C2 \& B4$ blocks while in the second choice, $D2, D3, \& D4$ are used to realize block $A3$ but the $B4$ block to realize $A1$. The second case option represents an alternative in which certain lower level blocks are installed in the design, together with higher level ones which can be built by these of lower level (as $B4$ is installed in the design along with $D2, D3, \& D4$ which themselves can implement the functionalities of $B4$). You can see that in the GNG of the second choice, $B4$ is duplicated. The first $B4$ is installed in the design to achieve the functions of $A1$ (where $FS(B4) = \phi$ as you can see in ★ on the right side of Fig. 3.2). The second $B4$ is built up by the installed $D2, D3$, & $D4$ blocks to perform the $A3$ functionality, in which case $B4$ is uninstalled in the design so $FS(B4)$ in not empty (see ★★ on the right side of Fig. 3.2).

However, notice that the GNG of the first option is a G-path with root $\Im$ while that of the second is not. The generated graphs of options resembling to the second choice always have a duplicated part, which contradicts to the illustration of a subdirected-hypergraph and thus don't correspond to G-paths. Consequently, the options of implementation can be split into those whose generated graphs are G-paths with root $\Im$, and those in which duplication occurs in their generated graphs.

3.1.3 Describing the Multi-Standard Directed Hypergraph from Mono-Standard Directed Hypergraphs

The graph representation of the multi-standard SDR system can be deduced from those of the single standards. Theoretically, we can explain how we can plot the directed hypergraph representation of the multi-standard system from the directed hypergraphs representing the break-down of each of the supported standards separately. We will use the following as an example to clarify things, in which we deduce

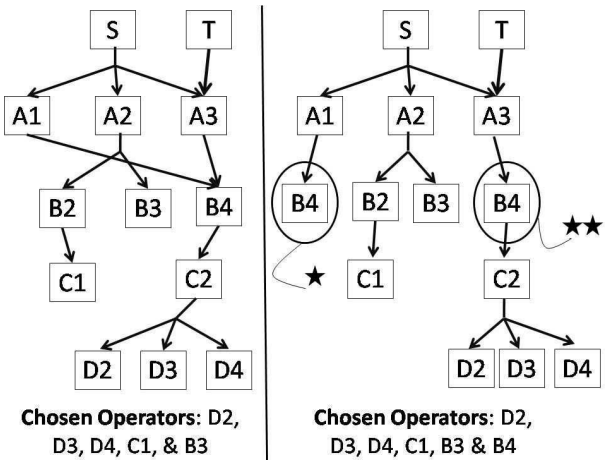

Figure 3.2: The generated graphs (obtained from Fig. 3.1) of two different options of implementation

the graph structure of the multi-standard system supporting both Wifi & UMTS, from the graph structure of each of Wifi and UMTS alone.

A graphical representation related to each of the two standards Wifi and UMTS is shown in figures 3.3 and 3.4 respectively. They are partial versions of the complete figures with only few important building blocks from transmitter side only.

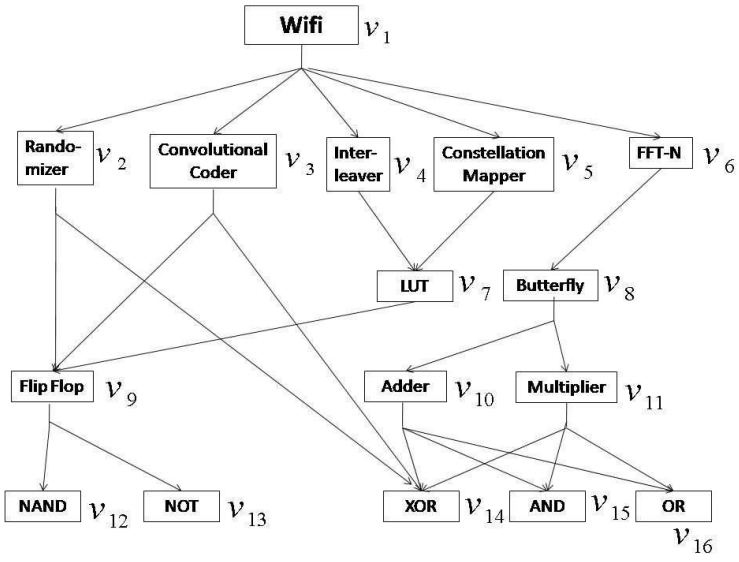

Figure 3.3: Global graphical structure for Wifi standard - transmitter side (up to 4 lower levels break-down) with the labeling of the vertices.

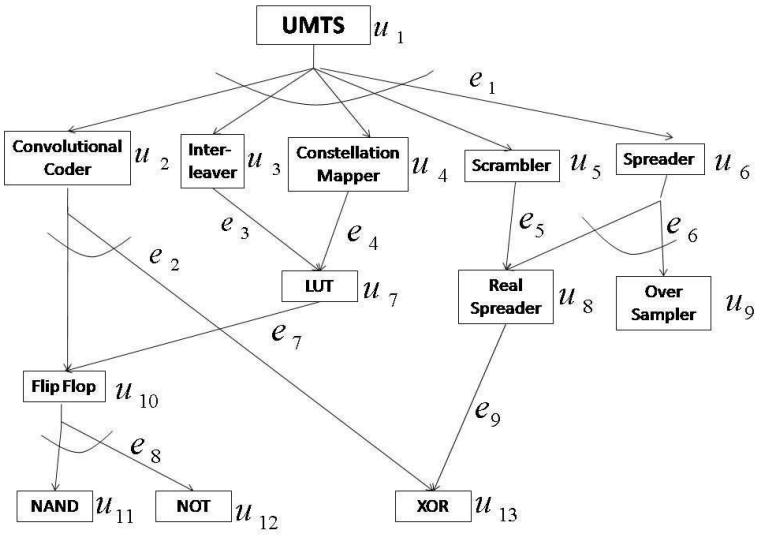

Figure 3.4: Global graphical structure for UMTS standard - transmitter side (up to 4 lower levels break-down) with the labeling of the vertices and the hyperarcs

Let $T_1 = (V(T_1), E(T_1))$ be the directed hypergraph representing all the alternatives to implement a single standard S_1 (as an example let S_1 be Wifi of Fig. 3.3) and $T_2 = (V(T_2), E(T_2))$ be the directed hypergraph that illustrates the different alternatives to implement another single standard S_2 (let S_2 be UMTS of Fig. 3.4 as an example). Our objective is to represent graphically all the alternatives capable of implementing the multi-standard terminal supporting both S_1 and S_2. Let T be this directed hypergraph that illustrates the multi-standard implementation alternatives supporting S_1 and S_2.

Most probably, there are some operators that can be used in the implementation of both standards. Accordingly, most likely, $\exists\, V \subseteq V(T_1)$ and $\exists\, U \subseteq V(T_2)$ such that:

$$V = U \text{ in the operator's sense}$$

When we say "in the operator's sense" this means by considering the elements of U and V as functional operators and functions like FFT, adder, multiplier operators etc.
For example, Convolutional Coder is an operator that can be used in the implementation of both standards Wifi and UMTS, it is vertex v_3 in $V(T_1)$ (of Fig. 3.3) and vertex u_2 in $V(T_2)$ (of Fig. 3.4) so we write $v_3 = u_2$ in the operator's sense.
Similarly we have: $v_4 = u_3$, $v_5 = u_4$, $v_7 = u_7$, $v_9 = u_{10}$, $v_{12} = u_{11}$, $v_{13} = u_{12}$, $v_{14} = u_{13}$ in the operator's sense.
So we have:
$V = \{v_3, v_4, v_5, v_7, v_9, v_{12}, v_{13}, v_{14}\}$, & $U = \{u_2, u_3, u_4, u_7, u_{10}, u_{11}, u_{12}, u_{13}\}$.
and we write $V = U$ in the operator's sense.

We define the function $f : V \longrightarrow U$ such that
$\forall\, v \in V \quad f(v) = u \Leftrightarrow v = u$ in the operator's sense.
f is a bijective mapping.

Definition 7. Let $e \in E(T_2)$. Define the hyperarc e' such that:

- $\forall\, r \in T(e) \cap U (resp.\ r \in H(e) \cap U),\ \ f^{-1}(r) \in T(e^{'})$ (resp. $f^{-1}(r) \in H(e^{'})$)
- $\forall\, r \in T(e) \backslash U$ (resp. $r \in H(e) \backslash U$), $\quad r \in T(e^{'})$ (resp. $r \in H(e^{'})$).

For example, let's find $e_1^{'}$ and $e_2^{'}$ for hyperarcs e_1 and e_2 of Fig. 3.4. According to definition 7 we get:
$e_1^{'} = (\{u_1\}, \{v_3, v_4, v_5, u_5, u_6\}) \quad \& \quad e_2^{'} = (\{v_3\}, \{v_9, v_{12}\})$

In the multi-standard hypergraph T, the common operators (between $V(T_1)$ and $V(T_2)$) which are the elements in the sets U and V will only be introduced once, and all hyperarcs in $E(T_1)$ and $E(T_2)$ which use a certain common operator will share it. Consequently the directed hypergraph T will be defined by the couple $(V(T), E(T))$ such that:

- $V(T) = V(T_1) \cup (V(T_2) \backslash U)$.
- $E(T) = E(T_1) \cup \{e^{'}\ / e \in E(T_2)\}$.

Fig. 3.5 represents the Wifi-UMTS multi-standard SDR directed hypergraph which can be deduced from the recent definitions of $V(T)$ and $E(T)$.

Once T is defined and given a third directed hypergraph T_3 that illustrates all the alternatives that are able to implement a single standard S_3, we can, in an analogous way, draw a directed hypergraph $T^{'}$ which represents the different alternatives capable of implementing all the three standards S_1, S_2 *and* S_3.
Thus by induction, we can define a directed hypergraph for a multi-standard terminal supporting any number of standards.

3.1.4 A formal cost function equation

Consider again the cost function introduced in chapter 1, which evaluates the cost of any selected option of implementation, duplicated below:

$$CF = (\sum_i BC_i.N_i + \sum_n \sum_k CC_k((S_n)_{n \in \{1,2 \cdots, N\}})) \tag{3.3}$$

In this part, we are going to derive an alternative theoretical formal expression of this cost function using definitions and notations of directed hypergraphs [88].

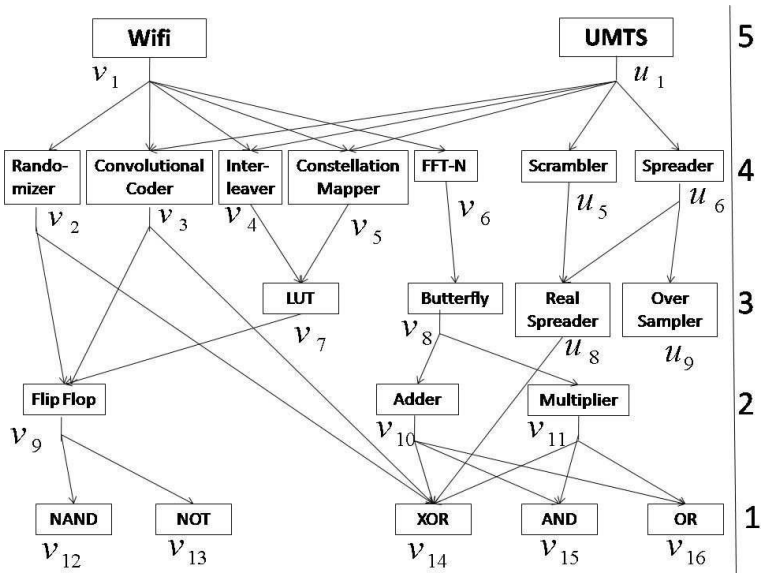

Figure 3.5: Global structure of the graph for multi standards (supporting Wifi and UMTS) - transmitter side (up to 4 lower levels break-down of the 2 standards) with the labeling of the vertices and their levels on the right of the figure.

For all what follows, we will use the term weight to mean the NoCs associated to any BF-reduction of a hyperarc in the directed hypergraph. This means that the number of times block x calls block y (which will be a number attached on the BF-reduction $(\{x\},\{y\})$ of a certain hyperarc e) will be denoted by $w_e(x,y)$. Sometimes for simplicity, we write $w(x,y)$ instead of $w_e(x,y)$ when the hyperarc e is clear or if it's the only one which admits such a BF-reduction.

Consider again the first choice of implementation in 3.1.2 (choosing the operators $D2, D3,$
$D4, C1, \&B3$) to implement the standards S and T of Fig. 3.1, whose GNG is pictured on the left side of Fig. 3.2. Then the cost of implementation via this choice (according to equation 3.3) is calculated as follows:

$$\text{Cost} = (((CC(D2)\times w(C2,D2)+CC(D3)\times w(C2,D3)+CC(D4)\times w(C2,D4))\times w(B4,C2))\times w(A3,B4))\times w(S,A3)+(((CC(D2)\times w(C2,D2)+CC(D3)\times w(C2,D3)+CC(D4)\times w(C2,D4))\times w(B4,C2))\times w(A1,B4))\times w(S,A1)+((CC(C1)\times w(B2,C1))\times w(A2,B2)+CC(B3)\times w(A2,B3))\times w(S,A2)+(((CC(D2)\times w(C2,D2)+CC(D3)\times w(C2,D3)+CC(D4)\times w(C2,D4))\times w(B4,C2))\times w(A3,B4))\times w(T,A3)+BC(D2)+BC(D3)+BC(D4)+BC(C1)+BC(B3)$$

$$= (((1\times 10+2\times 20+3\times 30)\times 4)\times 5)\times 8+(((1\times 10+2\times 20+3\times 30)\times 4)\times 4)\times 6+((10\times 5)\times 2+11\times 3)\times 7+(((1\times 10+2\times 20+3\times 30)\times 4)\times 5)\times 3+10+5+6+100+150$$

If we expand and factorize the above factors we'll get:

$= CC(D2)\ [w(C2,D2) \times w(B4,C2) \times w(A3,B4) \times w(S,A3) + w(C2,D2) \times w(B4,C2) \times w(A1,B4) \times w(S,A1)] + CC(D3)\ [w(C2,D3) \times w(B4,C2) \times w(A3,B4) \times w(S,A3) + w(C2,D3) \times w(B4,C2) \times w(A1,B4) \times w(S,A1)] + CC(D4)\ [w(C2,D4) \times w(B4,C2) \times w(A3,B4) \times w(S,A3) + w(C2,D4) \times w(B4,C2) \times w(A1,B4) \times w(S,A1)] +$
$CC(C1)\ [w(B2,C1) \times w(A2,B2) \times w(S,A2)] + CC(B3)\ [w(A2,B3) \times w(S,A2)] + CC(D2)\ [w(C2,D2) \times w(B4,C2) \times w(A3,B4) \times w(T,A3)] + CC(D3)\ [w(C2,D3) \times w(B4,C2) \times w(A3,B4) \times w(T,A3)] + CC(D4)\ [w(C2,D4) \times w(B4,C2) \times w(A3,B4) \times w(T,A3)] + BC(D2) + BC(D3) + BC(D4) + BC(C1) + BC(B3)$

$$= CC(D2)\ [\sum_{P\,:\,SD2-\text{path}} w(P)] + CC(D3)\ [\sum_{P\,:\,SD3-\text{path}} w(P)] + CC(D4)\ [\sum_{P\,:\,SD4-\text{path}} w(P)] + CC(C1)\ [\sum_{P\,:\,SC1-\text{path}} w(P)] + CC(B3)\ [\sum_{P\,:\,SB3-\text{path}} w(P)] + CC(D2)\ [\sum_{P\,:\,TD2-\text{path}} w(P)] + CC(D3)\ [\sum_{P\,:\,TD3-\text{path}} w(P)] + CC(D4)\ [\sum_{P\,:\,TD4-\text{path}} w(P)] + BC(D2) + BC(D3) + BC(D4) + BC(C1) + BC(B3)$$

$$= \sum_{P\,:\,SD2-path} CC(D2) \times w(P) + \sum_{P\,:\,SD3-path} CC(D3) \times w(P) + \sum_{P\,:\,SD4-path} CC(D4) \times w(P) + \sum_{P\,:\,SC1-path} CC(C1) \times w(P) + \sum_{P\,:\,SB3-path} CC(B3) \times w(P) + \sum_{P\,:\,TD2-path} CC(D2) \times w(P) + \sum_{P\,:\,TD3-path} CC(D3) \times w(P) + \sum_{P\,:\,TD4-path} CC(D4) \times w(P) + BC(D2)+BC(D3)+BC(D4)+BC(C1)+BC(B3)$$

$$= \sum_{x/\,d^{+}_{GNG}(x)=0} \sum_{P\,:\,Sx-\text{path}} CC(x) \times w(P) + \sum_{x/\,d^{+}_{GNG}(x)=0} \sum_{P\,:\,Tx-\text{path}} CC(x) \times w(P) + \sum_{x/d^{+}_{GNG}(x)=0} BC(x)$$

.

So generally, we can write the cost function (CF) as follows:

$$CF = \sum_{y/d^{-}_{GNG}(y)=0} \Big(\sum_{x/d^{+}_{GNG}(x)=0} \sum_{P\,:\,yx-\text{path}} CC(x) \times w(P) \Big) + \sum_{x/d^{+}_{GNG}(x)=0} BC(x) \tag{3.4}$$

where:

- $\sum_{x/d^{+}_{GNG}(x)=0} BC(x)$ represents the total sum of BCs of the blocks x such that $d^{+}_{GNG}(x) = 0$, which stand for the installed blocks in the design.

- $\sum_{P\,:\,yx-\text{path}} CC(x) \times w(P)$ is the necessary CC imposed by the installed block x responsible for realizing the standard y (y is a highest level block standard since $d^-_{GNG}(y) = 0$).
- $\sum_{x/d^+_{GNG}(x)=0} \sum_{P\,:\,yx-\text{path}} CC(x) \times w(P)$ stands for the total CC imposed by all the installed PEs x in the design to perform the functionality of the standard y.
- $\sum_{y/d^-_{GNG}(y)=0} (\sum_{x/d^+_{GNG}(x)=0} \sum_{P\,:\,yx-\text{path}} CC(x) \times w(P))$ represents the total CC paid for all the standards.

The calculation process of the above cost function is as follows: select a top level standard y and one installed block x, then search for all paths P (in the GNG associated to the choice of implementation) from y to x (i.e yx-paths) in order to multiply the weight of each such paths by the CC of the installed block x. Repeat the same operation for each y standard and x installed block. In this way, we get the total CC of the system. Finally, we have to add the BC of each installed block.

In the next section, we will provide an exploration of the total number of options capable of implementing the functionalities of the multi-standard system. More precisely, we will find an upper bound for this number, providing and explaining all the necessary derivations.

3.2 An upper bound for the number of options of implementation

In this section, we will provide an upper bound for the total number of options that are capable of implementing the multi-standard system [89]. This is done by first introducing a computational cost vector X_v on each vertex v in the graph structure of the multi-standard system, whose dimension ($\dim(X_v)$) will represent the total number of options that can realize block v. Then, we find an upper bound for $\dim(X_v)$, thus achieving our goal.

3.2.1 The computational cost vector X_v

We'll associate a vector X_v with each vertex v in the directed hypergraph H of a multi-standard system containing L levels. Each entry of X_v will represent the total CC resulting from one particular choice of implementation chosen to realize block v, where this cost is calculated via the cost function of equation 3.3. This vector will include all the possible implementations of v and thus the dimension of X_v will be exactly equal to the total number of options capable of realizing v.

For all the rest, we'll denote the dimension of the vector X_v by $|X_v|$, i.e $|X_v| = dim(X_v)$. The parameters that we need to form the entries of the X_v vector will be the BC, CC and the NoCs.

The vector X_v is defined recursively from the lowest level blocks up until those of highest levels. This means that first we have to find X_v for all v block in level 1, then X_v for all v block in level 2, $\cdots$.
For each block v in level 1 ($d^+(v) = 0$ in this case), we have only one entry in X_v (i.e $|X_v| = 1$), since the only choice of implementation of such a block is by installing it. This only entry in X_v will be the CC of block v, which will represent the total CC imposed when the block is installed by itself.
Having defined X_v for all the blocks v such that $L(v) \leq i$, the vector X_v where $L(v) = i + 1$ is formed as follows:

- If we face an "or" hyperarc $e \in FS(v)$ (recall that an "or" hyperarc means that $|H(e)| = 1$) and suppose that $H(e) = \{r\}$ (so $e = (T(e), H(e)) = (\{v\}, \{r\})$), then this means that v can be realized by r associated with certain number of calls. Since r can be implemented in $|X_r|$ ways, then this will impose $|X_r|$ choices capable of realizing v by using r because the total CC of any option that implements r, multiplied by the number of times v calls r ($w(v, r)$), represents the total CC of an option that realizes v. In other words, any entry in X_r multiplied by $w(v, r)$ will be an entry in X_v describing the total CC of one of the options of implementation of v via r.

- If an "and" hyperarc $e \in FS(v)$ is encountered, then v will need the functionalities of all the blocks in $H(e)$ to be implemented. Suppose that $H(e) = \{s_{i1}, s_{i2}, \cdots, s_{in}\}$ (so $e = (T(e), H(e)) = (\{v\}, \{s_{i1}, s_{i2}, \cdots, s_{in}\})$). In this case, the calculation of the total computational cost of an option chosen to implement v via this hyperarc e will be:
  choose one entry from each of $X_{s_{ik}}$, $k \in \{1, 2, \cdots, n\}$ (which represents the total CC of one of the options that can realize the functionalities of s_{ik}), multiply it by the number of times v calls s_{ik} (which is $w(v, s_{ik})$), and then add all the answers obtained for all $k \in \{1, 2, \cdots, n\}$. This will form an entry of X_v, because this is the way how the total CC of an option that can realize v will be calculated when facing an "and" connection (as indicated in equation. 3.3). Consequently, it's obvious that this hyperarc imposes $|X_{s_{i1}}| \times |X_{s_{i2}}| \times \cdots \times |X_{s_{in}}|$ options capable of realizing v.

- One more option of implementation which is worth mentioning is characterized by using v itself as a unified nondivisible block. This adds one more entry to the vector X_v which is the CC of block v.

To make things clearer, we'll start with a simple example before providing a generalization.

3.2.1.1 Example

Fig. 3.6 shows an "or" ($e_1 = (\{v_5\}, \{v_1\})$) and an "and" ($e_2 = (\{v_5\}, \{v_2, v_3, v_4\})$) connection related to the implementation of block v_5.

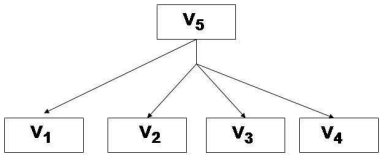

Figure 3.6: two alternatives to implement v_5

Suppose that:

$$X_{v_1} = (x_1, x_2, \cdots, x_m) \ ; \ |X_{v_1}| = m$$
$$X_{v_2} = (y_1, y_2, \cdots, y_n) \ ; \ |X_{v_2}| = n$$
$$X_{v_3} = (z_1, z_2, \cdots, z_p) \ ; \ |X_{v_3}| = p$$
$$X_{v_4} = (l_1, l_2, \cdots, l_q) \ ; \ |X_{v_4}| = q$$

We'll denote Z_v by the multiset obtained from X_v by just listing its components. So, for example, we have $Z_{v_1} = \{x_1, x_2, \cdots, x_m\}$, $Z_{v_2} = \{y_1, y_2, \cdots, y_n\}$, $\cdots$.

Set $U_{e_1} = Z_{v_1}$ (related to the "or" connection)
and $U_{e_2} = Z_{v_2} \times Z_{v_3} \times Z_{v_4}$ (concerning the "and" connection).

In this case, X_{v_5} will be a vector of dimension $m + n \times p \times q + 1$ where:

- The m entries which result from U_{e_1} are:
  $w(v_5, v_1) \times x_i \quad \forall i = 1, \cdots, m.$

- The $n \times p \times q$ entries which result from U_{e_2} are:
  $w(v_5, v_2) \times a + w(v_5, v_3) \times b + w(v_5, v_4) \times c \quad \forall (a, b, c) \ \in U_{e_2}$
  (where $|U_{e_2}| = n \times p \times q$).

- The remaining entry is the result of the direct implementation of block v_5 itself. This entry, as mentioned before, is equal to the CC of block v_5.

3.2.1.2 Generalization

Let H be a directed hypergraph of a multi-standard system and $v \in V(H)$. $\forall$ $e \in FS(v)$, set $U_e = \prod_{r \in H(e)} Z_r$.
The components of X_v will be found as follows:

- $\forall e \in FS(v)$, $\forall a = (a_r)_{r \in H(e)} \in U_e$ we have: $\sum_{r \in H(e)} w(v,r).a_r$ is an entry in X_v.
- One more entry of X_v is the CC of v which, as mentioned before, presents the direct installation of block v itself.

3.2.2 An upper bound for $|X_v|$

The dimension of X_v, $|X_v|$: Based on all what's explained and discussed previously in this section, we can easily conclude that :

$$|X_v| = \sum_{e \in FS(v)} \prod_{r \in H(e)} |Z_r| \; + \; 1 \tag{3.5}$$

defined recursively from lowest to highest levels.

We'll denote u_i by an upper bound for $|X_v|$, for any v block occupying the i^{th} level, i.e: $\forall\, v \;/\; L(v) = i, \quad |X_v| \leq u_i$. An expression of u_i will be our desired upper bound. u_i will be, as well, defined recursively from lowest to highest levels.
The following two parameters will be used:

$$M = \max_{v \in V(H)} (|FS(v)| + 1).$$

$$r = \max_{e \in E(H)} (|H(e)|).$$

We can deduce a recurrence relation as follows:

$$\begin{cases} u_1 = 1, \\ \\ u_{s+1} = M(u_s)^r \quad \forall\, 1 \leq s < L \end{cases} \tag{3.6}$$

Here is a brief explanation. Suppose that we want to find u_s, which is an upper bound for the number of options that can realize block v or an upper bound for $|X_v|$, where $L(v) = s$. One can notice the following remarks to obtain the recurrence relation of 3.6:

- We have a maximum of $M - 1$ hyperarcs such that v is the parent tail node (by the definition of M).
- Each one of these hyperarcs contains a maximum of r head nodes.
- The worst case is that all the r head nodes of a hyperarc $e \in FS(v)$ are in level $s - 1$, which will impose a larger upper bound.

We can conclude from equation 3.5 that each one of these (maximum) $M - 1$ hyperarcs imposes a maximum of $(u_{s-1})^r$ options, and thus the $M - 1$ hyperarcs all together will yield a maximum of $(M-1)(u_{s-1})^r$ options. It remains to add the last option of implementation characterized by using the block v by itself. So as a whole,

we get a maximum of $(M-1)(u_{s-1})^r + 1$ alternatives of implementation which is obviously less than or equal to $M(u_{s-1})^r = u_s$.

Thus we can obtain the following:

$$u_1 = 1$$
$$u_2 = M(u_1)^r = M$$
$$u_3 = M.(u_2)^r = M(M)^r = M^{r+1}$$
$$u_4 = M.(u_3)^r = M(M^{r+1})^r = M^{r^2+r+1} \dots$$

Notice that the recurrence relation of equation 3.6 can be easily solved with a simple induction on s and we obtain: $u_s = M^{r^{s-2}+r^{s-3}+\cdots+r+1}$ or alternatively:

$$u_s = M^{\frac{1-r^{s-1}}{1-r}}; \;\; s > 1, \;\; r \neq 1 \tag{3.7}$$

As an example, we will find the exact number of options to implement the multi-standard system of Fig. 3.1 as well as our attained upper bound for this number. Applying equation 3.5, we can deduce that:
$|X_{D1}| = |X_{D2}| = |X_{D3}| = |X_{D4}| = |X_{D5}| = |X_{C1}| = |X_{C3}| = |X_{B3}| = 1$, $|X_{C2}| = 3$, $|X_{B1}| = |X_{B2}| = 2$, $|X_{B4}| = 5$, $|X_{A1}| = |X_{A3}| = 6$, $|X_{A2}| = 10$, $|X_S| = 361$, $|X_T| = 7$. Consequently, the exact number of options capable of implementing the multi-standard design consisting of the standards S and T will be $|X_S| \times |X_T| = 361 \times 7 = 2527$.

In this same figure, we can notice that $M = \max_{v \in V(H)} (|FS(v)| + 1) = 4$ and $r = \max_{e \in E(H)} (|H(e)|)$ $= 3$, and thus $u_5 = 4^{\frac{1-3^5-1}{1-3}} = 4^{40}$ is an upper bound for the total number of options to implement each of S and T blocks in level 5. Consequently $u_5 \times u_5 = 4^{40} \times 4^{40}$ is our attained upper bound for the number of options to implement the multi-standard design.

In this section we have found an exponential upper bound, as function of the selected parameters M and r, for the total number of options that can implement any PE in the design. Consequently, we derived an upper bound for the total number of alternatives capable of implementing all the top level standards to be supported, and thus have presented an exploration of the number of options that can realize the multi-standard design. In the following section, we will exhibit our optimization problem that states to find one of these options of implementation having a minimum cost. This problem is expected to be a complex problem due to the attained exponential upper bound for the total number of alternatives of a multi-standard design. In this context, the complexity of this optimization problem will be further examined and elaborated.

3.3 Complexity of our optimization problem

Our objective is to optimize the cost function introduced in 3.1.4 to its minimum cost possible and thus solving the optimization problem that finds balance between efficiency and flexibility. This will enable us to extract the most appropriate COs from the most convenient granularity levels leading to the construction of an optimal SDR multi-standard design which takes advantage of the common aspects in use. In this section, we will give a formal description of our optimization problem and study its complexity [90].

Our optimization problem can be described as follows:

A general description of the parameters: The parameters of an instance of our problem must represent a graph structure H of a multi-standard system. The following is a list of the variables of our optimization problem.

1. a list of the n vertices in $V(H)$.
2. a list of the m hyperarcs in $E(H)$.
3. the number of levels L.
4. the number of blocks in each level. Let $a_1, a_2, \cdots, a_L$ denote the number of blocks in levels $1, 2, \cdots, L$ respectively.
5. A list of the highest level blocks (occupying the L^{th} level) which are required to implement.
6. The numerical values "CC & BC" of each block, the NoCs on each arc (which are the weights on the BF-reductions of all the hyperarcs).

Statement: Find the set of operators $U \subseteq V(H)$ which is capable of implementing the multi-standard terminal and that has a minimum cost.

Recall that any optimization problem, whether a minimization or a maximization one, can be transformed into a decision problem (yes-no question) by just introducing a new constant parameter $B \geq 0$ (see section 2.5). Accordingly, the decision problem version of our optimization problem will be stated as follows:

The general description of the parameters:

- points 1 to 6 in the parameters' description of the optimization problem.
- a constant $B \geq 0$.

Statement: Can we find a set of operators $U \subseteq V(H)$ which implements the design and whose cost doesn't exceed B?

Recall again that a decision problem is said to be an NP-problem if any "yes"-instance of the problem can be decided in polynomial time by a Nondeterministic Turing Machine (NDTM). An NDTM is said to operate in "polynomial time" if there exists a polynomial p s.t, for every yes-instance I, there is some guess S that leads the deterministic checking stage to respond "yes" for I and S within time $p(Length[I])$, where $Length$ is a function which maps an instance

I to the number of symbols in the string by which it's represented.

In an equivalent way to the Turing Machine's proof process, one can prove that a decision problem is an NP-problem if for a given "yes" instance I and a certain guess S, a polynomial equation of the required operations can be derived to verify that the answer for I and S is "yes". All the details can be found in chapter 2. We will exploit this idea in the following theorem to prove that our decision problem version is an NP-problem but under a specific stated condition. Consequently, the optimization problem will be an NP-problem under the same condition as well, since it's at least as hard as its decision problem version.

Theorem 10. *The previously described decision problem is an NP-problem on condition that the number of levels L of a multi-standard graph structure is upper bounded by a non-negative constant i, i.e $L \leq i$ for some $i \geq 0$.*

Proof. In our proof, we will follow the previously explained strategy. We will consider an instance I of our problem and a certain guess solution S for this instance, then try to derive a polynomial equation for the number of operations required to check if the answer for I and S is "yes". This imposes considering worst case situations which require maximum number of operations. However, since it's generally impossible to determine an exact worst case, we are going to determine scenarios that are worse than the worst case.
Consider any graph structure H of a multi-standard system with all the necessary parameters (instance I), containing L levels. Guess a certain solution S (a solution to our problem is a set $U \subseteq V(H)$) and suppose that $|U| = k$ (i.e we have chosen some k blocks randomly), we will have to check the following three points:

- First, verify if the guess S can implement the design:
  We propose one way for doing this. Let $e \in E(H)$. If $H(e) \subseteq U$, then update U by adding $T(e)$ to U. In fact, if all the head nodes of e are in U, then the blocks in U are capable of realizing the parent node in T(e) accompanied by the necessary number of calls. Consequently, the functionality of the "T(e)" node can be attained and is thus added to the set U to mean that this corresponding block can be built by the elements in U. This is applied to all the hyperarcs in $E(H)$.

  The worst case is to consider that the k chosen blocks are in the lowest level (level 1) and thus passing by all the hyperarcs in $E(H)$ once might, in the worst case, form some blocks in level 2 (or more but we're considering the worst case). Now U is updated in the way explained before (new blocks might be added to U). Passing by all the hyperarcs another time and considering the updated U set shall, in the worst case as well, form some blocks in level 3 which have to be added to U. Clearly, we need to check the m hyperarcs in $E(H)$ at most $L-1$ times in order to reach the functionalities of the highest level blocks in level L.

After the maximum of $m(L-1)$ searches in the hyperarcs, if all the top level standards were in the latest updated set U, then this will mean that the initial guess of $U \subseteq V(H)$ is a choice which can implement the design. Otherwise, the guessed solution will be considered incapable of implementing the multi-standard system.

As a conclusion, we need a maximum of $m(L-1)$ operations to check this point.

- Second, calculate the cost of the chosen option which is characterized by the selected blocks in U:
  In this step, we have to find the number of multiplications and number of additions required in the calculation process of the cost function. A GNG resembling to that in fig. 3.7, call it W_{GNG}, corresponds to a worst choice of implementation because:
  1. The k chosen blocks occupy the lowest level (level 1) which yield the longest paths from the top level standards to the chosen blocks in level 1, thus maximizing the number of multiplications when finding the weight of the paths.
  2. An "and" connection between any block v and all the blocks which occupy a lower level than v corresponds to a worst case, since it imposes a maximum number of paths from the standards to the chosen installed blocks in U.

  In our case, we are sure that a practical realization of an SDR multi-standard system won't be more complex than the case of Fig. 3.7. That's why we can consider it to be worse than the worst case.

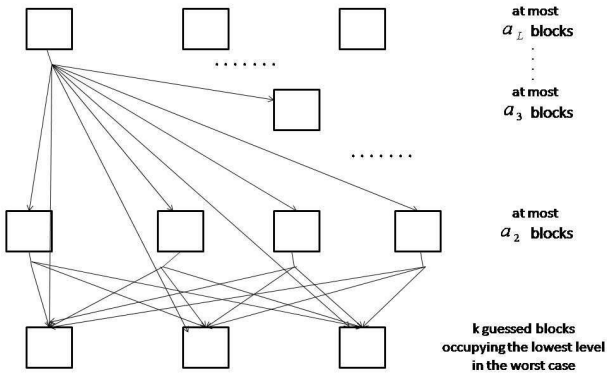

Figure 3.7: W_{GNG}, a worst case GNG of a choice of implementation.

In the following, we will explore the total number of paths in W_{GNG} from $\Im$ to a vertex in level 1.

Let v be a vertex in W_{GNG} with $L(v) = i$. We can always find a path from any vertex in $\Im$ to v traversing any combination and any number of vertices from levels $L-1$ till $i+l$ (probably no vertex at all) all occupying different levels than each other. Note that the paths traverse the vertices in a decreasing order of the level that each vertex occupies.

Obviously, all the nodes in W_{GNG} occupying a certain level are connected to $\Im$ by the same number of paths. Let v be a vertex in level i. We'll denote n_i by an upper bound for the number of paths from $\Im$ to v.

We can easily remark that a vertex in level $L-1$ is connected to $\Im$ by at most a_L paths in W_{GNG}, thus $n_{L-1} = a_L$.
Let y be a vertex in level $L-i$ in W_{GNG}. The value n_{L-i} is equal to $n_{L-(i-1)}(a_{L-(i-1)}) + n_{L-(i-1)}$. In fact, each path from a vertex u in $\Im$ to a vertex x in level $L-(i-1)$ in W_{GNG} (where the total number of such paths is at most $n_{L-(i-1)}$) can be extended to a path from u to y via a hyperarc $E \in FS(x) \bigcap BS(y)$. This makes at most $n_{L-(i-1)}(a_{L-(i-1)})$ paths, traversing all the vertices x in level $L-(i-1)$ in W_{GNG}. The remaining paths which don't traverse a vertex in level $L-(i-1)$ are at most $n_{L-(i-1)}$ paths, since any path from $\Im$ to a vertex x in level $L-(i-1)$ in W_{GNG} can be transformed into a path from $\Im$ to y by just replacing the destination x of each path by y.

So, we can attain the following recurrence relation:

$$\begin{cases} n_{L-1} = a_L \\ \\ n_{L-i} = n_{L-(i-1)}(a_{L-(i-1)}) + n_{L-(i-1)} \end{cases} \tag{3.8}$$

Let $s = \max\{a_2, \cdots, a_L\}$. By a simple induction process, we can easily conclude that n_{L-i} belongs to $O(s^i)$. Consequently $n_1 = n_{L-(L-1)}$, which stands for an upper bound of the total number of paths from $\Im$ to one vertex in level 1 in W_{GNG}, belongs to $O(s^{L-1})$. Note that the total number of paths from $\Im$ to all the k installed blocks in level 1 is at most kn_1.

The calculation of the weight of each of these kn_1 paths is required, according to the cost function of equation 3.4. A maximum of $L-1$ multiplications is needed to calculate the weight of any one of these paths, because the longest of these paths is of length $L-1$. One more multiplication is associated to each of these paths since we have to multiply its weight by the CC of the corresponding installed block. So, we can conclude that each path is associated with at most L multiplications. Consequently, the maximum number of multiplications required will be at most kLn_1.

As for the maximum number of additions imposed by the cost function, we need one addition between the weight of a path and another. Since there exists a maximum of kn_1 paths, then we'll get $kn_1 - 1$ number of additions

in the worst case.

Thus as a whole, the total number of multiplications and additions necessary to calculate the cost will be less than: $kLn_1 + (kn_1 - 1)$.

- The third and last necessary point is to compare the cost (found in the second step) with B. This will just be a matter of one operation.

Finally, the total number of operations which are required to verify a "yes" answer for the instance I and the guess S will be:

$$m(L-1) + [kLn_1 + kn_1 - 1] + 1. \tag{3.9}$$

If we set b to denote $max\{s, k, m\}$, then we get that the number of operations obtained in equation 3.9 is a function in $O(b^L)$ (recall that $n_1 \in O(s^{L-1})$ and $L \leq i$). In this case where a non-negative constant i upper bounds L (as hypothesized in the theorem), we'll get that equation 3.9 is a function in $O(b^i)$, thus requiring polynomial time.

□

3.4 Conclusions

First in this chapter we have explored various theoretical aspects concerning the SDR multi-standard design, from which we mention modeling the graph structure of a multi-standard system as a directed hypergraph and deriving a formal cost function equation. Afterwards, we have provided an upper bound for the total number of options capable of realizing the multi-standard design. The fact that this attained upper bound is exponential was a clue that our optimization problem, which states to select the minimum cost option capable of implementing the design, is an NP-problem. This is true on condition that the number of levels in the graph structure of the SDR multi-standard system doesn't exceed a certain constant, which usually is a logical condition from the technical point of view.

Since we now know that the optimization problem we're dealing with is a complex problem, then one idea is to try to examine the different alternatives of implementation of an SDR multi-standard system in order to exclude or disregard some of them which have certain identified characteristics, if possible. This point is studied in the next chapter before we propose a new algorithm, which exploits various definitions and notations concerning directed hypergraphs. This algorithm checks each possible option of implementation but ignores some of them, and then returns the minimum cost option of implementation. The difference between our algorithm and the previously suggested ones for this problem will further be highlighted.

Chapter 4

An Optimization technique for Multi-Standard SDR equipment using Directed Hypergraphs

Contents

In the previous chapters, we presented a suggested cost function equation. Our objective is to optimize the proposed cost function to its minimum value possible, using graph theoretical tools, by choosing the most adequate option of implementation having the minimum cost. We showed in chapter 3 that this optimization problem is an NP-problem if the number of levels doesn't exceed a certain constant, thus concluding that this is a complex problem. Consequently, this problem is related to determining an optimal or near-optimal resource sharing for multi-standard systems which is faced with a complex objective cost function.

We start this chapter in section 4.1 with an overview of some state-of-the-art previous elaborated optimization techniques to solve our optimization problem, where the best chosen method was one which gives a near optimal solution. Afterwards in this chapter, we study the characteristics of the different op-

tions of implementation and prove in 4.2 that some specific alternatives can be excluded if one is searching for a minimum cost design. We propose a new algorithm in section 4.3 to solve the optimization problem, using different proposed modeling notions related to directed hypergraphs, which only examines the unexcluded alternatives of implementation in section 4.2. The significance of this algorithm lies in the fact that, on the contrary to heuristic techniques, it is capable of providing an exact optimal solution. A worst-case analysis of this algorithm is further presented in 4.4, which yields an upper bound on the resources required by the algorithm. Finally, we apply our suggested algorithm on several generic design examples in 4.5 in order to explore its performance, before we end this chapter with a conclusion's section.

4.1 Some state-of-the-art techniques of optimization

To determine a solution to our optimization problem, one has the choice between techniques that give an exact-optimal solution or those that provide a near-optimal solution in less computing time. All exact methods for determining an optimal solution require a computing effort which increases exponentially with the instance's size (if one is dealing with intractable problems), so that in practice exact solutions can be attempted only on instances with small size. We start in this section with a brief overview of one exact-optimal and another near-optimal technique, which were among the examined techniques by a previous PhD student in [60] to solve our optimization problem.

Exhaustive search (ES) is a technique which finds the best global solution after checking each and every solution in the search space, i.e it provides an exact-optimal solution. It is the simplest technique in its implementation; it only requires to generate every possible solution to the problem systematically. In the case of high-dimensional, NP-complete, or multi-modal, $\cdots$ etc problems, this method becomes impractical or even infeasible as the search space becomes larger. Since the size of the search space of real world problems is usually enormous, then it requires centuries or more to find optimal solutions for such problems. Thus, for intractable problems, ES can only be applied to instances with a relative small size.

Simulated annealing (SA) [103] is one of the most applicable heuristics, that provides near-optimal solutions, in the optimization community. It is a stochastic search strategy which exploits the analogy between the way in which metal cools and freezes into a minimum energy crystalline structure and the search for a minimum in a more general system.
In fact, Metropolis *et al.* [104] proposed an algorithm to find the equilibrium configuration of a collection of atoms at a given temperature, but the connection between this algorithm and mathematical minimization was first noted in [105]. However, it was Kirkpatrick *et al.* [106] who proposed that this algorithm forms the basis of an optimization technique for many problems.

SA algorithm employs a random search which not only accepts changes that decrease the objective function value, but also some changes that increase it. The latter are accepted with a certain probability $p < 1$ which exponentially decreases either with time or with the amount by which the current optimum is worsened [107]. It's this property which makes SA powerful, i.e its ability to escape from being trapped in local minima by accepting worse moves through a probabilistic procedure especially in the earlier stages of the search. If the metal temperature is decreased slowly enough, then metal cools and freezes into a minimum energy crystalline strucure; the analogy for SA is that if the move probability decreases slowly enough, then the global optimum is attained.

To solve our optimization problem, S. Gul [60] has applied some selected optimization techniques to a small generic example and finally deduced that SA was the best among them, since it finds the optimum solution in less number of iterations than the rest. He validated the results of the selected stochastic techniques by comparing them to the results obtained by ES, and that's why he was restricted to a simple case example which contains few nodes (less than 10). Certainly, he couldn't have chosen the ES technique as a tool of optimization for our problem because of its exponential growth in terms of number of iterations and consequently the time required with the increase in the considered PEs. Moreover, S. Gul was intuitively convinced that the problem is a complex one, but the proof of this was not done till now, which can be found in section 3.3 of chapter 3 of this thesis.

However, S. Gul was aware of the fact that SA need not necessarily be the best possible optimization technique, but this selection perfectly met his needs for the different case studies that he presented in his thesis. In our work, we need to find new tools of optimization with the use of graph theory. It would have been a good idea if we could have found a modeling of our optimization problem to a previously graph-theoretical solved one (like shortest path problem, MST problem, $\cdots$ etc) but our problem was kind of different and more complicated and thus this wasn't possible to our best knowledge. So, we have created our own modeling and solution for this problem. The first step was the graphical approach of the SDR multi-standard system modelized as a directed hypergraph as well as the derivation of a formal theoretical expression of the proposed cost function, presented in chapter 3. Since indeed we have proved that our optimization problem, which states to minimize the cost function, is an NP-problem (under a certain constraint), then we thought of trying to examine the options of implementation to check if some of them can be ignored in order to reduce the complexity. This will constitute the work presented in the next section.

4.2 Excluding certain designs when searching for the one with minimum cost

It has been highlighted in 3.1.2 of chapter 3 that there are certain alternatives of implementation of a multi-standard system in which a certain block is installed along with some others which can build it, all in the same design. In this section, we're going to prove that such designs whose generated graphs don't correspond to G-paths of the graph structure of the SDR multi-standard system, don't have the minimum cost [90]. We'll prove this first on an example and then provide a generalization. In both cases, three designs will be exhibited. Design (1) will represent a GNG obtained by realizing the multi-standard system using higher level blocks than in the case of design (2), where lower level blocks will be used to realize all the above necessary functions which they are capable of building. On the contrary, design (3) will exploit the lower level blocks to implement a certain function but the higher ones (which can be built by these low level ones) to perform some other function. The aim is to prove that design (3), whose GNG admits a duplicated part, will never correspond to a design with minimum cost.
In our demonstration, first we'll assume that cost (design (2)) doesn't exceed cost (design (1)) and accordingly, prove that cost (design (2)) will be less than cost (design (3)). As a result, design (2) will be the cheapest among the three in this case. The other possibility that will need a proof is to suppose that design (2) is more expensive than both designs (1) & (3), in which case design (1) will be found to be a cheaper design than the third, thus proving that the first design is the cheapest. Consequently, we conclude that in all possible cases, either the first or second design will have the minimum cost among the three designs. This will be enough to support our claim that the cost of a third design case can never be minimum and that we can always find a design which has a lower cost.

4.2.1 An example

An example is pictured in Fig. 4.1. In concurrence to what's mentioned right before, design (1) represents a design in which the implementation of the blocks D and F is achieved by the high level blocks H & I while in design (2), the lower level blocks $J, K, L,$ & I will be installed to realize both D and F. As for the third design, the choice is to implement F using $J, K, L,$ & I while to use the higher level blocks H & I (which can be implemented by the lower level blocks $J, K, L,$ & I) to realize the functionalities of D.
Note that all the three designs represent the GNG of a certain choice of implementation and are obtained from some directed hypergraph representation of an SDR multi-standard system.

Remark that in design(3) of Fig. 4.1, there is a Duplicated Part (DP) (including the A & H functions) in which one of them is traversed by one of the paths

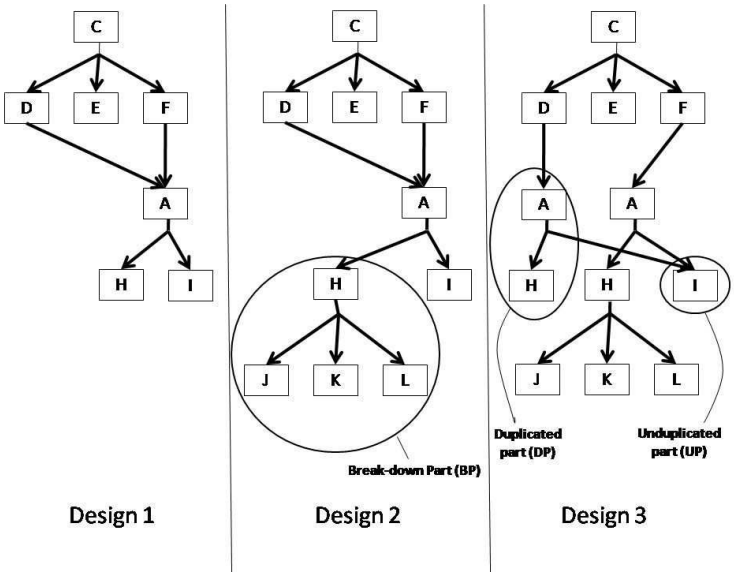

Figure 4.1: A 3 designs example

needing its functionalities (the path passing through D). The second path reaching the DP (passing through F) considers further break-down of some of its functions (see the Break-down Part (BP) of the H processing element in Fig. 4.1).
Inside the DP of Fig. 4.1, only one highest level block exists which is A. Note that there is an Unduplicated Part (UP) (the I function) which will be sharing in the CC of the highest level block A of the DP.

In fact, the only difference between the three designs lies in whether further breakdown of the DP is considered or not. Later in this subsection, we will demonstrate that among the three designs in Fig. 4.1, either design(1) or design(2) can have the minimum cost but never design(3), where the cost is evaluated via the cost function introduced in chapter 1 (whose alternative is presented in chapter 3).
Remark the following:

$$\begin{aligned} CC(A)_{\text{(with no further break-down)}} &= CC(A)_{\text{(using } H \text{ \& } I\text{)}} \\ &= CC(H) \times w(A,H) + CC(I) \times w(A,I) \end{aligned} \tag{4.1}$$

$$\begin{aligned} CC(A)_{\text{(with further break-down)}} &= \text{CC(A)}_{\text{(using } J,K,L \text{ \& } I\text{)}} \\ &= (\text{CC(J)} \times w(H,J) + CC(K) \times w(H,K) \\ &\quad + CC(L) \times w(H,L)) \times w(A,H) + CC(I) \times w(A,I) \end{aligned} \tag{4.2}$$

In both equations 4.1 and 4.2, there is a part sharing in the $CC(A)$ imposed by the UP, which is $CC(I) \times w(A, I)$. The other part in 4.1 (denoted by a) will be sharing in the $CC(A)$ imposed by the DP **with no** further breakdown, while that in 4.2 (denoted by r) will be sharing in $CC(A)$ imposed by the DP **with** further breakdown, i.e we have:

- $a = CC(H) \times w(A, H)$
- $r = (CC(J) \times w(H, J) + CC(K) \times w(H, K) + CC(L) \times w(H, L)) \times w(A, H)$

Now, we'll calculate the cost of each design in Fig. 4.1. We'll denote Cost(design i) by Cost(i). Then we get:

- $Cost(1) = (a + CC(I) \times w(A, I)) \times w(F, A) \times w(C, F) + (a + CC(I) \times w(A, I)) \times w(D, A) \times w(C, D) + CC(E) \times w(C, E) + BC(E) + BC(H) + BC(I)$
- $Cost(2) = (r + CC(I) \times w(A, I)) \times w(F, A) \times w(C, F) + (r + CC(I) \times w(A, I)) \times w(D, A) \times w(C, D) + CC(E) \times w(C, E) + BC(E) + BC(J) + BC(K) + BC(L) + BC(I)$
- $Cost(3) = (r + CC(I) \times w(A, I)) \times w(F, A) \times w(C, F) + (a + CC(I) \times w(A, I)) \times w(D, A) \times w(C, D) + CC(E) \times w(C, E) + BC(E) + BC(H) + BC(I) + BC(J) + BC(K) + BC(L)$

The procedure followed to prove that the third design can never be the minimum cost design has been lately presented.
The first case proof is as follows:
Suppose we were in the case where Cost(2) $\leq$ Cost (1). Then, we will prove that Cost(2) is strictly less than Cost(3).

Assume to the contrary that Cost(3) $\leq$ Cost(2).
$\Longrightarrow \boxed{\begin{array}{c}(a + CC(I) \times w(A, I)) \times w(D, A) \times w(C, D) + BC(H) \leq \\ (r + CC(I) \times w(A, I)) \times w(D, A) \times w(C, D)\end{array}} \bullet$
$\Longrightarrow (a + CC(I) \times w(A, I)) \times w(D, A) \times w(C, D) < (r + CC(I) \times w(A, I)) \times w(D, A) \times w(C, D)$ (because $BC(H) > 0$)
$\Longrightarrow a + CC(I) \times w(A, I) < r + CC(I) \times w(A, I)$ (the weights are always positive).
$\Longrightarrow \boxed{a < r.}$ (1)

But we have Cost(2) $\leq$ Cost(1).
$\Longrightarrow (r + CC(I) \times w(A, I)) \times w(F, A) \times w(C, F) \leq (a + CC(I) \times w(A, I)) \times w(F, A) \times w(C, F) - BC(J) - BC(K) - BC(L) +$
$\underbrace{[(a + CC(I) \times w(A, I)) \times w(D, A) \times w(C, D) + BC(H)]}$
$\underbrace{- [(r + CC(I) \times w(A, I)) \times w(D, A) \times w(C, D)]}_{\leq 0 \text{ by } \bullet}$
$\Longrightarrow (r + CC(I) \times w(A, I)) \times w(F, A) \times w(C, F) < (a + CC(I) \times w(A, I)) \times w(F, A) \times w(C, F)$ (since $BC(v) > 0 \quad \forall$ v block)

$\Longrightarrow \boxed{r < a}$ (2) contradiction of (1)

Thus, the assumption that Cost(2) $\geq$ Cost(3) is proved to be wrong leading to the conclusion that Cost(2) is in fact less than Cost(3). So in this case, a minimum cost design choice will be the second one.

Suppose we are in a different case where Cost(2) > Cost(1) & Cost(3) > Cost(2). Under these inequalities, it's obvious that among the three designs, design(1) has the least cost, which makes it the minimum cost design in this case.

Now consider the last possibility, where we are given that Cost(2) > Cost(1) but Cost(3) $\leq$ Cost(2). We will prove that Cost(3) is certainly strictly greater than Cost(1), thus proving that the minimum cost design in this case will be the first design form. Actually, the first condition (Cost(2) > Cost(1)) will be useless in this proof because in fact, the condition Cost(3) $\leq$ Cost(2) is alone sufficient to conclude that Cost(3) exceeds Cost(1).

Suppose the contrary, i.e Cost(3) $\leq$ Cost(1).
$\Longrightarrow (r + CC(I) \times w(A, I)) \times w(F, A) \times w(C, F) < (a + CC(I) \times w(A, I)) \times w(F, A) \times w(C, F)$
$\Longrightarrow \boxed{r < a}$ (3)
And we have Cost(3) $\leq$ Cost(2)
$\Longrightarrow \boxed{a < r.}$ (4-done before in 1) which contradicts inequality 3.

So, you can notice that in all possible cases, either design(1) or design(2) is the minimum cost design and there is no possibility that the third one admits the least cost.

4.2.2 Generalization

In this subsection, we will generalize our study of the possibilities for designing a minimum cost design. The duplicated and unduplicated parts (denoted by DP and UP) speak of their names, where indeed the previous example illustrates a pictorial view for each one of them. Recall that the DP and UP can only be recognized from the third design form. We will further need to consider the following two notations:

- DUP denotes the combination of both DP and UP.
- DUPB is a combination of the DUP functions along with the break-down part (BP) of some of the operators in DP.

For example in the case of Fig. 4.1, we can remark that DUP consists of the blocks A, H & I while A, H, I, J, K, & L form those in DUPB. Note that H & I are the lowest level blocks in DUP; J, K, L & I are the lowest level blocks in DUPB.
The three designs are generally illustrated as in Fig. 4.2. .

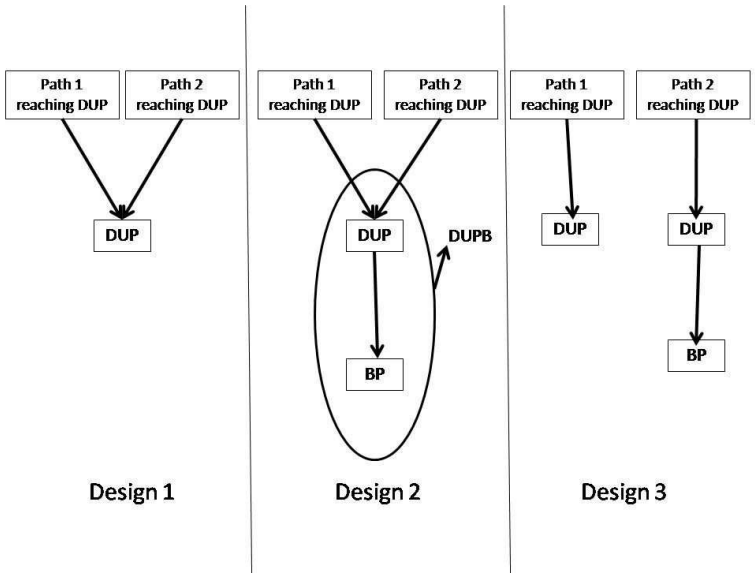

Figure 4.2: The three designs generalization case

In an analogous way to the example in 4.2.1, design (1) consists of two paths both realized by the lowest level blocks in DUP, while the second design (design (2)) includes two paths both implemented by the lowest level blocks in DUPB. As for design (3), one of the two paths is implemented using the high level installed functions from DUP, and the other is realized by the lower level installed operators from DUPB. We will prove the following theorem.

Theorem 11. *An option of implementation of an SDR multi-standard system whose GNG admits a duplicated part can never have the minimum cost.*

Proof. To prove this theorem, it's enough to demonstrate that design(3) of Fig. 4.2 doesn't correspond to a minimum cost design and that we can always find another design possessing a lower cost.
Suppose that the DP contains one highest level block inside it, block A for example. Recall the following notations:

- $a = CC(A)$ imposed by the DP **with no** further break-down.
- $r = CC(A)$ imposed by the DP **with** further break-down.

As in 4.2.1, the $CC(A)$ can be divided into 2 parts, one imposed by the DP and the other by the UP as follows:

- $CC(A)_{(with\ no\ further\ break-down)} = CC(A)$ imposed by the DP **with no** further break-down $+ CC(A)$ imposed by the UP
  $= a + CC(A)$ imposed by the UP.
- $CC(A)_{(with\ further\ break-down)} = CC(A)$ imposed by the DP **with** further break-down $+CC(A)$ imposed by the UP
  $= r + CC(A)$ imposed by the UP.

Since the DP and UP can contain any number of blocks distributed on different possible levels, then it's impossible to provide a precise equation of the cost of each of the three designs as was done in the previous example, where the DP and UP were a concrete distribution of blocks. However, we can easily conclude that when we settle comparison inequalities between the costs of two design forms, the cost imposed by the UP cancel from both sides of the inequalities. Afterwards, factorizing some costs originating from the paths reaching DUP and DUPB leaves us with only expressions comparing between the parameters r and a. After having understood these concepts which were detailed in the previous example, it will be easy to conclude what follows.

Suppose that Cost(2) $\leq$ Cost(1). Then we get Cost(2) $<$ Cost(3) because if we assume to the contrary that Cost(3) $\leq$ Cost(2) we get $a < r$, when in fact Cost(2) $\leq$ Cost(1) implies that $r < a$ thus leading to a contradiction.

The remaining possibility to consider is to suppose that Cost(2) $>$ Cost(1) & Cost(3) $\leq$ Cost(2) and then conclude that Cost(3) exceeds Cost(1) in this case. In fact, a contradiction can be attained since the assumption that Cost(3) is less than or equal to Cost(1) yields that $r < a$ while Cost(3) $\leq$ Cost(2) implies that $a < r$.

Now, suppose that the DP contains more than one highest level block, two as a beginning (for instance, the DP in Fig. 4.3 includes two highest level blocks $A\&B$). We will tackle a discussion to prove that the theorem holds in this case as well.

In fact, we will split the DP into several DPs in which each one of them contains exactly one highest level block, and then work on each DP separately. For example, the DP $(A, C, D, B, \&E)$ in Fig. 4.3 can be split into two different DPs, (A, C, D) & (B, E), in which the former includes only one highest level block which is A and similarly for the latter in which the only highest level block is B.
We can consider first the break-down of A in the three design forms while fixing in all of them a certain form of the break-down of B. Three break-down forms of B exist, where all the three cases are successively considered. In each of the three cases, the DP will only have one highest level block A, thus returning to the case presented in the beginning of this subsection. Accordingly, we can conclude that we should consider the break-down of A based on either the first or second design form when searching for a minimum cost design no matter how was the break-down of B. Again, performing the same operation on the break-down of B in the 3 design forms while fixing in all of them a certain form of the break-down of A will yield the same result, that is B should be broke-down either with a design similar to design (1) or (2) when a designer seeks a minimum cost system no matter how was the break-down of A.

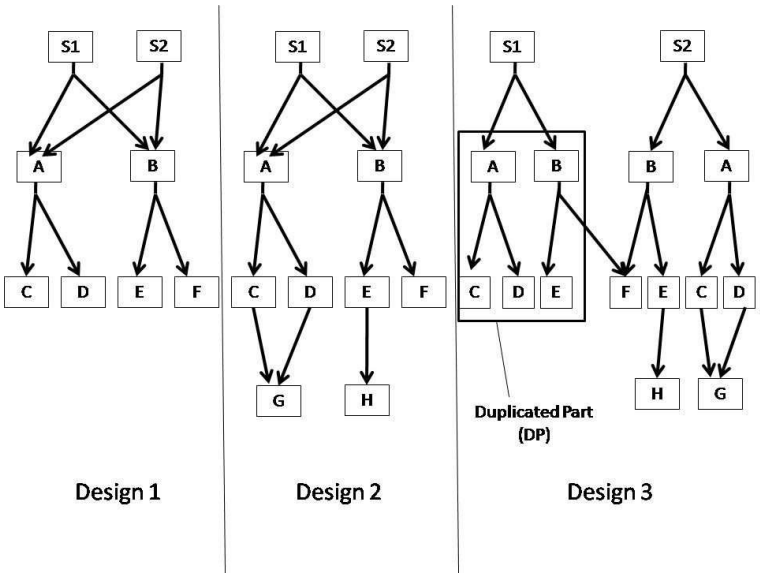

Figure 4.3: Example of three designs case where the DP contains two highest level blocks

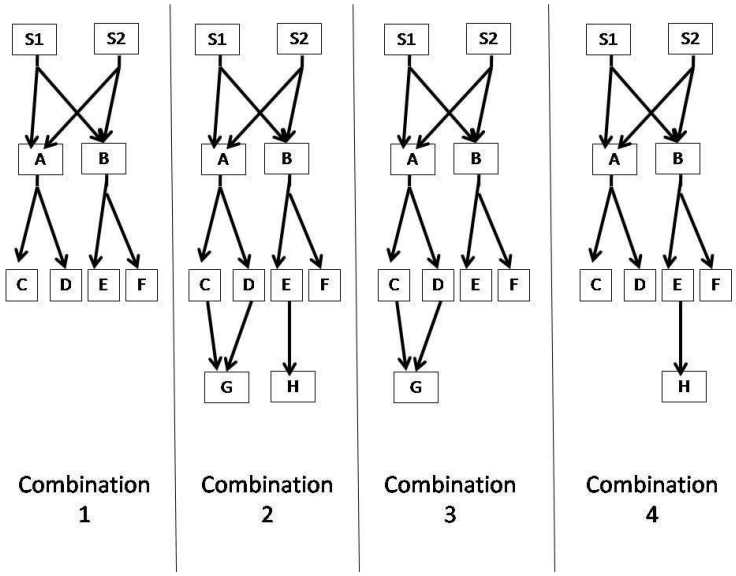

Figure 4.4: The four combinations of break-down of $A\&B$ that should be tested when searching for minimum cost designs, thus excluding the rest which include duplication of either the break-downs of A and/or B

As a conclusion, we can remark that there will be four combinations, illustrated in Fig. 4.4, that should be examined if we seek a design with minimum cost. The first combination involves using the break-downs of A and B, both based on the first design form (**with no** further break-down of both $A\&B$), while the second combination illustrates the break-down of A and B based on the

second design form (**with** further break-down of both $A\&B$). As for the two remaining combinations, they are both formed such that one of the blocks (A or B) is realized with no further break-down, while the other block (A or B) is implemented by considering further break-downs.
We can recognize that the third design case of Fig. 4.3 can be excluded with complete safety if one seeks a minimum cost design, as this case is not among the four considered possibilities of Fig. 4.4.
By a simple induction process, we can extend the proof to a DP containing any number of highest level blocks, starting from 3 blocks and above.

This completes our proof. As a conclusion, we can say that a designer searching for a minimum cost design can ignore all those in which duplication occurs in their corresponding generated graphs.

□

Notice that the generated graphs of design(1) & design(2) are G-paths with root $\Im$ while that of the third design is not. This is because, as highlighted before in chapter 3, the GNG of options resembling to the third design form admit a duplicated part, and thus are not subdirected-hypergraphs of the SDR multi-standard's graph structure.
We can remark that the options of implementation can be split into those whose generated graphs are G-paths with root $\Im$, and those in which duplication occurs. However, we have just proved in this section that the options which admit a duplication part can't in any way correspond to minimum cost designs and that we can always find ones having lower costs. Consequently, since our aim is to find the design with minimum cost, where the cost is calculated via the cost function presented in 3.1.4 in chapter 3, then we can restrict our study to the options whose generated graphs are G-paths with root $\Im$. This idea will be exploited in the next section, where we will propose a new algorithm that can solve our optimization problem by only examining each option whose generated graph is a G-path of root $\Im$, instead of checking all the possible options of implementation. It's an algorithm which exploits the different proposed modeling notions related to directed hypergraphs.

4.3 A Minimum Cost Design (MCD) Algorithm

In this section, we will propose an algorithm which is capable of solving our optimization problem [102]. We will restrict our search to the options that are illustrated as G-paths with root $\Im$, which we proved in the previous section to be enough when seeking minimum cost designs.

Let H be a directed hypergraph representing the break-down of a multi-standard SDR system. We will define H_r, the directed hypergraph obtained from H, as follows:

- $V(H_r) = V(H) \cup \{r\}$.

- $E(H_r) = E(H) \cup \{E_r\}$ where $E_r = (\{r\}, \Im)$.

In fact, H_r is obtained from H by adding an imaginary top level vertex r to $V(H)$ and the hyperarc E_r to $E(H)$, where $\{r\}$ is the tail set of E_r and $\Im$ is that of its head.
Since the vertex r plays the role of an imaginary highest level standard, so this changes the graph structure of the multi-standard system H, which possibly contains several top level blocks, into H_r which only contains one.

The parameters assigned to the entities of H_r will be:

- CC, BC and NoCs for the entities of the directed hypergraph H_r (on the blocks and the arcs) remain the same for all the similar entities of the directed hypergraph H.
- $w_{E_r}(r, v) = 1 \ \forall \ v \in \Im$

Remark that it will be unnecessary to assign neither a BC or CC to r, nor a level that it occupies since it is an imaginary block.

In our algorithm when calculating the cost of an option, we will need to search for all the paths in H from each of the standards in $\Im$ to all the installed blocks in a selected option. This will be equivalent to searching for all the paths from only the vertex r in H_r to the same installed blocks in the option, thus reducing some steps of the algorithm. Besides, note that the weights on all BF-reductions of the E_r hyperarc are set to 1, in order to ensure that a path from r to an installed block will have the same weight as that of the associated path from a certain vertex in $\Im$ to the same installed block. Consequently, the cost of the design will not be influenced by the insertion of the new vertex r.

Our algorithm is called the Minimum Cost Design (MCD) algorithm. The only input that it needs will be the directed hypergraph H_r obtained from H (where H represents the graph structure of a multi-standard system), together with the parameter entities of H_r. We will also need to enter the level of each block in H.

This algorithm will find all G-paths of H_r with root $\{r\}$ (representing certain options of implementation) in a step by step manner, generating options from others. It will compute the cost of every selected option to be tested (using the cost function explained in subsection 3.1.4 of chapter 3), compare its cost to the previously examined G-paths by the algorithm, and then generate several options (pictured as G-paths) that emerge from this selected one. The same procedure will be followed for any other selected option. Finally, it will exhibit as an output the G-path of H_r with root $\{r\}$ which has the minimum cost, together with its corresponding cost found. Once we have this G-path as output, we can extract the blocks which are to be installed in the design by just identifying the vertices with zero out-degree, thus achieving our goal of finding the most suitable common operators that will optimize the cost.

During the iterations of the algorithm we will introduce, for each selected G-path X, a vector k_v associated to every vertex $v \in V(X)$ defined recursively from the highest level nodes in X to the lowest. The dimension of this vector will be evaluated as follows:

$$\begin{cases} \dim k_v = 1; & v = r; \\ \dim k_v = \displaystyle\sum_{e \in BS_X(v)} \sum_{w \in T(e)} \dim k_w; & v \neq r; \end{cases} \tag{4.3}$$

Note that r will be the top level vertex of every G-path found.
The k_v vector can be denoted by $(k_{1v}, k_{2v}, k_{3v}, \cdots, k_{(\dim k_v)v})$, where each component of k_v will represent the weight of a path from r to v, and dim k_v will correspond to the number of such paths.

Furthermore in the algorithm, we will introduce a set Q in which the vertices of the option X in hand will be invoked gradually while traversing the associated algorithm's loops. However, a vertex to select at each step from Q will be that occupying the highest level among those present in Q, which was imposed by the recursive definition of the vectors k_v from higher level to lower level vertices. Thus, the algorithm selects an element u in Q at every step satisfying that: $L(u) = \max\{L(w); w \in Q\}$.

Many variables have been introduced in the algorithm. We will explain the benefit of some of them.

- M is a set in which the generated G-paths of H_r with root $\{r\}$ will be invoked.
- A variable R_P is introduced to occupy the total cost of the $p-th$ selected G-path.
- S is a variable in which we accumulate the cost of a certain G-path of H_r with root $\{r\}$.
- A is a set which will contain all vertices v in the selected G-path X with out-degree zero in X.
- SMin is an integer variable which will include the least cost of a G-path obtained so far.
- K is a variable in which we reserve the G-path of H_r with root $\{r\}$ having the least cost among those tested so far.

Here are the complete steps of the "MCD" algorithm:
Procedure$(H_r, CC(v), BC(v), NoC(v))$
begin
$M = \{(\{r\}, \phi)\}$, $R_p := 0$, $p := 1$, $K := \phi$;
repeat
  select and remove $X \in M$
  if $X = (\{r\}, \phi)$
    go to step U | **1**
  end-if
  $S := 0$, $A := \phi$;

```
k_r := k_1r := 1        dim k_r := 1;
for each v ∈ V(X)\{r} do
  k_v = 0 vector,     dim k_v := 0;          2
end-for
Q = {r};
repeat
select and remove v ∈ Q
for each E ∈ FS_X(v) do
begin
  for each h ∈ H(E) do                       3
  begin
    Q := Q ∪ {h}
    i := 1
    repeat
    if k_ih ≠ 0                              4
      i := i + 1
    end-if
    until k_ih = 0
    if v ≠ r
      for each E ∈ BS_X(v)
        for each w ∈ T(E)
          dim k_v := dim k_v + dim k_w        5
        end-for
      end-for
    end-if
    j := 0
    repeat
      k_(i+j)h = k_(j+1)v × w_E(v,h)          6
      j := j + 1
    until j := dim k_v
    if d_X^+(h) = 0
      l := i
      repeat
        S := S + CC(h) × k_lh                7
        l := l + 1
      until l = i + dim k_v
      A := A ∪ {h}
    end-if
  end-for
end-for
until Q = φ
repeat
select and remove v ∈ A                      8
  S := S + BC(v)
until A = φ
R_p := S
```

if $p = 1$ | **9**
  SMin $:= R_p$
  $K := X$
end-if
$p := p+1$ | **10**
$R_p := 0$
if $p > 2$ | **11**
  if $R_{p-1} <$ **SMin**
    SMin $:= R_{p-1}$
    $K := X$
  end-if
end-if
STEP U
for each $u \in V(X)$ **s.t** $d_X^+(u) = 0$ **do** | **12**
begin
  for each $E \in FS_{H_r}(u)$ **do**
    $M := M \cup \{X + E\}$
  end-for
end-for
until $M = \phi$
end-procedure.

Each of the 12 statements shown in the algorithm has its own significance. We'll briefly highlight the role of some of them.

1. Step 1 ensures that if $X = (\{r\},\phi)$ is the selected G-path from M, then we have to skip the calculation of the cost of X and go to Step U. This is because this G-path X has no technical significance for the implementation of the multi-standard SDR system, as r is an imaginary vertex added to H.
2. In step 2, as an initialization for the calculation of the cost of the G-path X, we fix $k_r = (1)$ and $k_v = 0$ vector $\forall\ v \neq r$. Note that k_v $(v \neq r)$ is a vector with unknown dimension at this step.
3. The role of step 4 is to identify the index i of the first zero component k_{ih} of vector k_h so that we can update it after, rather than updating on previously settled components of k_h.
4. Step 5 is the one in which we compute dim k_v, which will be required for the next steps. This computation follows the equations in 4.3. Remark that if $v = r$, then there will be no need to compute dim k_r, as it is initialized to 1 in step 2.
   Note that since in this step we calculate the dimension of the selected vertex v from Q, then all the entries of k_v should be already found and this is what imposed choosing the highest level block from Q every time we want to select a vertex. In fact, if we select a vertex v from Q which doesn't occupy the highest level among those in Q, it might happen that its vector k_v isn't fully completed, as for example a higher level unselected vertex t from Q might impose new entries in k_v (if $\exists\ E \in FS_X(t)$ and $v \in H(E)$).

5. The sixth step consists in multiplying all the components of k_v by $w_E(v,h)$ in order to obtain the weight of all the paths from r to h passing through v via the hyperarc E. However, we'll fill these new components of k_h starting from k_{ih}.
6. Step 7 is accessed only if $d_X^+(h) = 0$ (i.e if h is an installed block) obeying the calculation of the cost via the proposed cost function. In such cases, we multiply the newly calculated components of k_h in step 6 by $CC(h)$ and add each one of them in the variable S. Moreover, the vertex h is added to the set A.
7. After having calculated the total CC S of the G-path X, we still have to add the BCs of all the installed blocks (present in the set A) into S. This is achieved in step 8.
8. Step 9 consists in just assigning the cost "R_1" of the first selected option X in the variable SMin, with its corresponding G-path X assigned in K, as this will be the first and only option tested so far and thus will represent the option with the least cost.
9. The tenth step is responsible of initializing the next R_P to zero, in which we will be associating later the cost of the following G-path chosen from M.
10. In step 11 we update SMin to the possible lower cost found (if it was less) and update K to the corresponding lower cost G-path.
11. The final step generates G-paths of H_r from the G-path X. It searches for all vertices u with out-degree zero in X and for each, find the hyperarcs $E \in FS_{H_r}(u)$ in order to add the generated option $X + E$ to M. It can be easily concluded that such generated options are also G-paths.

A flowchart diagram of the MCD algorithm is pictured in Fig. 4.5.

The MCD algorithm clearly doesn't seem to be a fast algorithm due to the large number of instructions and loops it involves. In the following section, we will perform a worst case analysis to this algorithm. However, since it's typically impossible to determine an exact worst-case scenario (as in most worst case analysis problems), we will consider scenarios that are worse than the worst case. This computational complexity analysis will yield an upper bound on the resources required by the algorithm.

4.4 Computational Complexity of the MCD algorithm

Let H_r be an input directed hypergraph which contains L levels (disregarding the level of r as previously mentioned), with $|V(H_r)| = n$ and $|E(H_r)| = m$. We'll consider the following parameters:

- $E_i = \bigcup_{v/\ L(v)=L-i+1} FS(v)$; Obviously $E_L = \phi$.
- $t = \max_{i=1,\cdots,L-1} |E_i|$.
- $W = \{v / d_{H_r}^+(v) = 0\}$.

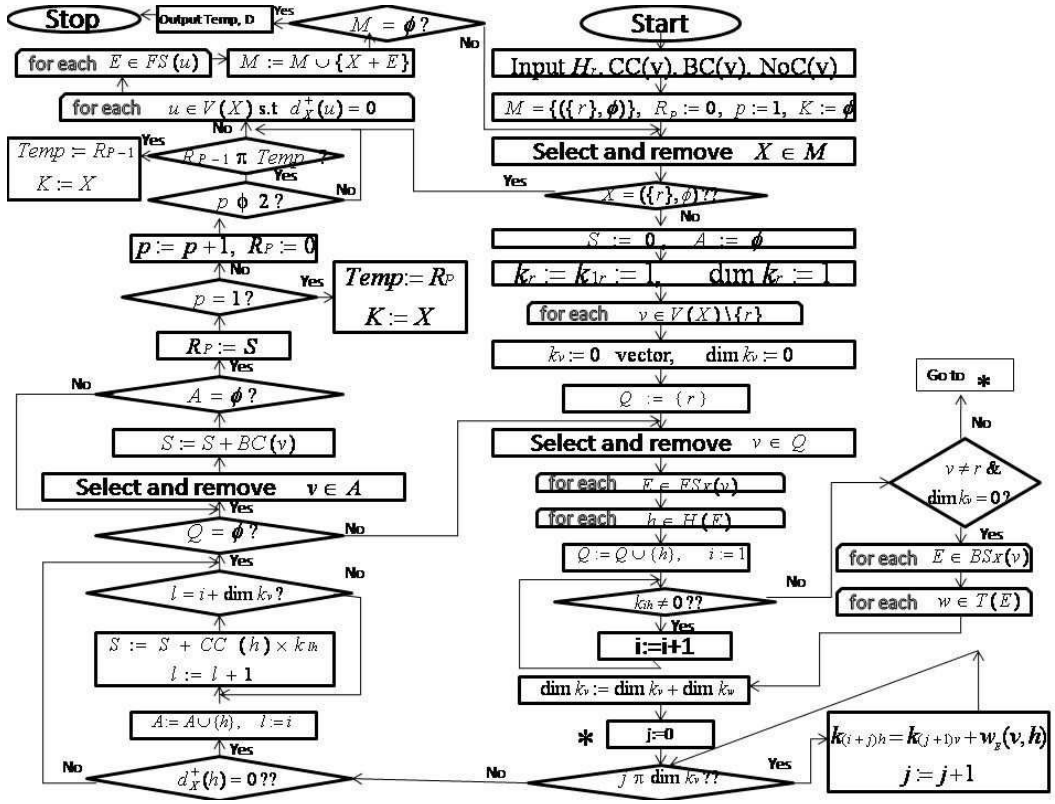

Figure 4.5: Structure of the Minimum Cost Design algorithm

A series of different steps are necessary to analyze the computational complexity of the MCD algorithm, which will be consecutively presented in this section [102].

4.4.1 The maximum number of hyperarcs in a G-path

We'll explore the maximum number of hyperarcs that any G-path of H_r with root $\{r\}$ can ever include. Let X be such a G-path. Since the out-degree of each vertex in X is either 0 or 1, and the tail set of any hyperarc in X is a non-empty set, then it can be easily concluded that $|E(X)| \leq |V(X)|$. As $|V(X)| \leq n$ (since $V(X) \subseteq V(H_r)$), so $|E(X)| \leq n$. More precisely, X contains at most $n - |W|$ hyperarcs because the vertices of out-degree zero in H_r will never contribute to add hyperarcs to any G-path.

4.4.2 An upper bound for the total number of G-paths

In this part, we will find an upper bound for the total number of possible G-paths of H_r with root $\{r\}$. For this, we adopt the same SDR option's generation concept described by the algorithm (step 12) to estimate this number of generated G-paths. As an initialization we have the option $X = (\{r\}, \phi)$, which

generates only one option from H_r defined by the G-path $(\{r\} \cup \Im, \{E_r\})$, and contains only one hyperarc. Obviously, this last option generates $|E_1|$ G-paths of H_r with root $\{r\}$, each of which contains exactly 2 hyperarcs. In turn, each of these $|E_1|$ options generates less than ct G-paths for some c constant, each containing exactly 3 hyperarcs. As an example, one of these $|E_1|$ options might generate less than $|E_1| + |E_2|$ G-paths with root $\{r\}$; another might form less than $|E_1| + |E_2| + |E_3|$, $\cdots$ depending on the existing levels of vertices in the selected G-path from the $|E_1|$ formed ones. As a conclusion, we can remark that the total number of generated G-paths of H_r with root $\{r\}$, which contain at most 3 hyperarcs each, will be a function in $O(t^2)$.

Proceeding in the same manner, we can remark that the total number of G-paths of H_r with root $\{r\}$ containing at most k hyperarcs will be a function which belongs to $O(t^{k-1})$. Consequently, the total number of options generated by the algorithm will be a function in $O(t^{n-|W|-1})$, since any G-path of H_r with root $\{r\}$ contains at most $n - |W|$ hyperarcs (explained in subsection 4.4.1).

4.4.3 An upper bound for the dimension of k_v

In some steps of our algorithm, we need to check all the components of a k_v vector for a certain task. In this subsection, we will find an upper bound for the dimension of such a vector in order to estimate the number of times we enter such loops.

Since the dimension of a k_v vector stands for the number of paths from the vertex r to v in a G-path with root $\{r\}$, we will have to investigate a vertex v which is connected to r by the maximum number of paths. Recall that a G-path with root $\{r\}$ is a directed hypergraph obtained from H_r whose vertices' out-degrees is either zero or one.

We'll introduce a particular G-path with root $\{r\}$, obtained from an input directed hypergraph H_r containing L levels, call it G_{max}. It is defined such that: it contains at least one vertex in level 1, the out-degree of each vertex v s.t $L(v) = i \quad (i \neq 1)$ is 1 where $FS(v) = \{E_v\}$ with $E_v = (\{v\}, \{x/L(x) < L(v)\})$, and the out-degree of r is 1 with $FS(r) = \{E_r\}$ (where $E_r = (\{r\}, \Im)$ as defined in 4.3). It's clear that the out-degree of the vertices in level 1 can't be except zero, like it's the case in any input directed hypergraph.
In other words, G_{max} is defined such that an "and" connection exists between any block $v \neq r$ and all the blocks which occupy a lower level than v. A pictorial view example of this particular defined G-path is illustrated in Figure 4.6. Note that the level of any vertex in a G-path of H_r with root $\{r\}$ is the same as that in H_r.

Let v be a vertex in G_{max} s.t $v \neq r$ and $L(v) = k$. A path from r to v traversing any combination and any number of vertices from levels L till $k + l$, all

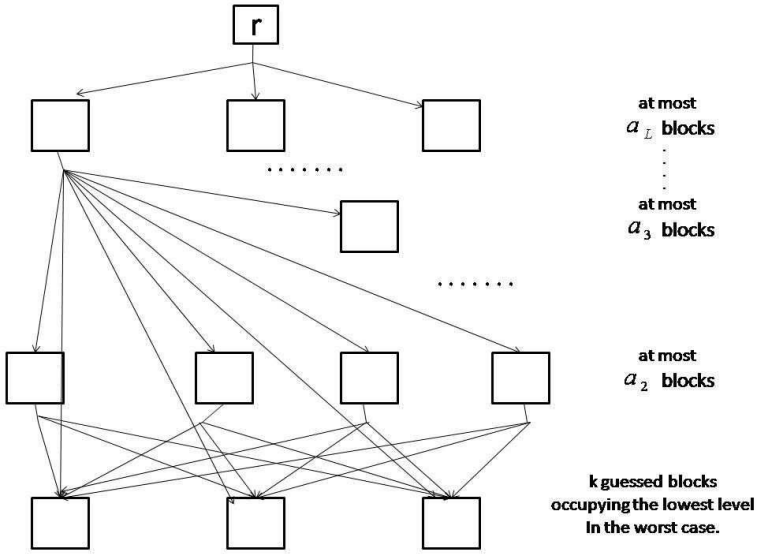

Figure 4.6: A particular illustration of the defined G-path of H_r with root $\{r\}$, "G_{max}"

occupying different levels than each other, can be found. Note that the paths traverse the vertices in a decreasing order of the level that each vertex occupies.

Obviously, all the nodes in G_{max} occupying a certain level are connected to r by the same number of paths. Let v be a vertex in level k. We'll denote an upper bound for the dimension of k_v in G_{max} by n_k and the number of vertices occupying level k in H_r by a_k.

It's clear that n_L is equal to 1, since any vertex x in level L in G_{max} can be reached from r by only 1 path which is (r, E_r, x). However, a vertex in level $L-1$ is connected to r by at most a_L paths, each one traversing one of the nodes in level L in G_{max}, thus $n_{L-1} = a_L$.
Let y be a vertex in level $L-k$ in G_{max}. The value n_{L-k} is equal to $n_{L-(k-1)}(a_{L-(k-1)}) + n_{L-(k-1)}$. In fact, a path from r to a vertex x in level $L-(k-1)$ (which are at most $n_{L-(k-1)}$ paths) can be extended to a path from r to y via a hyperarc $E \in FS(x) \bigcap BS(y)$. This makes an upper bound of $n_{L-(k-1)}(a_{L-(k-1)})$ paths, traversing all the vertices x in level $L-(k-1)$ in G_{max}. The remaining paths which don't traverse a vertex in level $L-(k-1)$ are at most $n_{L-(k-1)}$ paths, since the $n_{L-(k-1)}$ paths from r to a vertex x in level $L-(k-1)$ in G_{max} can be transformed into paths from r to y by just replacing the destination x of each path by y.

So we can write the following recurrence relation:

$$\begin{cases} n_L = 1 \qquad n_{L-1} = a_L \\ \\ n_{L-k} = n_{L-(k-1)}(a_{L-(k-1)}) + n_{L-(k-1)} \end{cases} \tag{4.4}$$

It's true that G_{max} doesn't represent a logical choice of implementation from the technical point of view but however, a vertex v occupying the first level in one of these G-paths can be connected to r by the largest number of paths, because all possible combinations of vertices to form a path from r to a vertex v in level 1 can be considered in G_{max}.

Let $s = \max\limits_{i=2,\cdots,L} a_i$. By a simple induction process, we can easily conclude that n_{L-k} belongs to $O(s^k)$. Consequently, $n_1 = n_{L-(L-1)}$ which stands for an upper bound of the dimension of any k_v vector ever, belongs to $O(s^{L-1})$.

Remark that a similar reasoning has been used in this subsection as well as in section 3.3 of chapter 3.

4.4.4 The worst case complexity analysis

We have showed in chapter 2 that $\sum_{v\in V(H_r)} d^-_{H_r}(v) = \sum_{e\in E(H_r)} |H(e)|$. In this part, we will denote both sums by d. Consider again our MCD algorithm for which we will find an expression of the computational complexity. It's clear that the cost of the initialization in step 2 of the algorithm is $O(n)$ time. We'll assume that each operation of selection and removal from sets M, Q, or A as well as insertion into M, Q, or A has unit cost.

Now, since each vertex is inserted and removed from the set Q at most once (because the selection of an element u from Q satisfies $L(u) = \max\{L(w); w \in Q\}$), we can conclude that in step 3 of our algorithm, each hyperarc in X will be examined only once (i.e the first time the hyperarc is selected) and at each time selected, all its head nodes will be examined in turn. Thus steps 4 till 7 of the algorithm will be executed $\sum_{e\in E(H_r)} |H(e)| = d$ times inside the loop "repeat $\cdots$ until $Q = \phi$".
Steps 4, 6 and 7 run in $O(s^{L-1})$ each time we enter the loop "repeat $\cdots$ until $Q = \phi$", as we need in all these steps to explore the components of a certain k_v vector, whose dimension is upper bounded by a function in $O(s^{L-1})$ (explained lately in subsection 4.4.3). On the contrary, step 5 will be executed a total of $\sum_{v\in V(H_r)} d^-_{H_r}(v) = d$ times all along the repeated iterations of the "repeat $\cdots$ until $Q = \phi$" loop, since the dimension of each vertex in X will be calculated only once.

Step 8 runs obviously in $O(n)$ time, as a maximum of n vertices exist in A. Steps 9, 10, and 11 require negligible execution, just a couple of assignments and comparisons. Step 12 runs in $O(m)$ time, as a maximum of m hyperarcs would be

added to the existing option in hand. All steps 1 till 12 run in $O(t^{n-|W|-1})$ time, since the loop "repeat $\cdots$ until $M = \phi$" is executed a "number of options" times, which is a function in $O(t^{n-|W|-1})$ (as derived in subsection 4.4.2).

Eventually, we can conclude after combining all the required time executions that the MCD algorithm runs in $O((ds^{L-1}+d+n+m)t^{n-|W|-1})$ or equivalently $O(dt^{n-|W|-1}s^{L-1})$ time. For a more compressed form, consider the parameter $a = \max(d, t, s)$. Then, the MCD algorithm runs in $O(a^{n-|W|+L-1})$ time. Thus, an exponential upper bound is attained for the resources required by the MCD algorithm.

This computational complexity analysis gives an idea that the time required by this algorithm might grow fast with the growth of the graph structure of the SDR multi-standard system, but still this is a worst-case analysis for the resources' requirments and need not necessarily represent the exact number in most of the cases. In the next section, we will exhibit some examples on which we will apply our algorithm, in order to highlight how its complexity actually might evolve.

4.5 Application

The MCD algorithm has been implemented by using C language. Details of the coding methods, functions and structures related to nodes, hyperarcs, directed hypergraphs, and other necessary definitions and relations can be found in the Appendix. In this section, we show the results of running this program code on several examples to give an idea of its performance, starting by one directed hypergraph example and evolving it by always adding one more hyperarc. For each one of these examples, we will state the attained output minimum cost design with its associated cost, the total number of options examined by the algorithm (i.e the total number of G-paths), and the execution time required to get the output.

Let's start with the directed hypergraph example of Fig. 4.7 which contains 26 nodes and 28 hyperarcs and on which we associate cost parameters with nodes and BF-reductions of all the hyperarcs. Recall that parameters associated with nodes are building cost/computational cost (BC/CC), while those associated to the BF-reduction of a hyperarc represent the number of calls (NoCs). It's to be noted that the values of all parameters are arbitrarily chosen. After entering this instance of a directed hypergraph to our program code, we got the following results:

- The minimum cost option of implementation is the one whose GNG is pictured in Fig. 4.8, characterized by installing in the design the operators $W, Y, K, \& R$.
- The cost of this option is 170.
- The number of G-paths examined by the algorithm are 99,190.

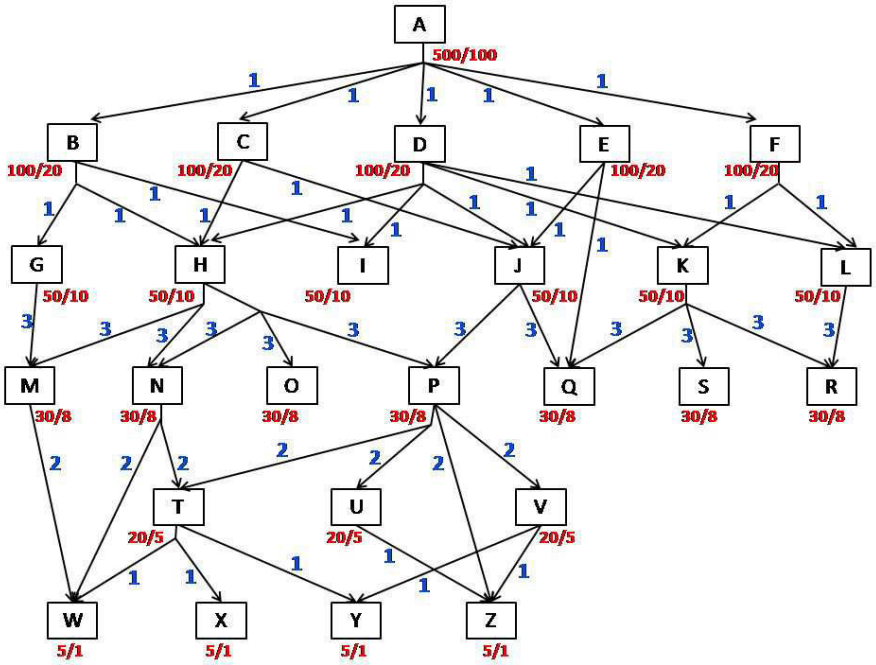

Figure 4.7: A directed hypergraph example containing 26 nodes and 28 hyperarcs

- The output was returned after about 95.8 minutes.

Note that the machine on which we were running our code has the following CPU and RAM characteristics: Genuine Intel(R) CPU with frequency 1.83 GHz, 2.00 GB of RAM.
The other directed hypergraphs examples on which we applied our algorithm were formed from the example of Fig. 4.7, each time adding one more hyperarc. We added consecutively the hyperarcs $(\{I\},\{S\})$, $(\{U\},\{X\})$, $(\{E\},\{K\})$, $(\{I\},\{O,P,Q\})$, $(\{M\},\{T\})$, and $(\{O\},\{T,U\})$ to form the examples illustrated in Figures 4.9, 4.10, 4.11, 4.12, 4.13, and 4.14 respectively. The minimum cost option of implementation selected by our program code was the same in all examples, pictured by the GNG of Fig. 4.8 and whose corresponding cost is 170. The number of options and time required for each one of these generic design examples are hereby stated.

Example 2 of Fig. 4.9:

- The number of options checked by the program code are 143,640.
- The output was returned after about 188 minutes.

Example 3 of Fig. 4.10:

- The algorithm checks 168,936 G-paths.
- The output was returned after about 258 minutes.

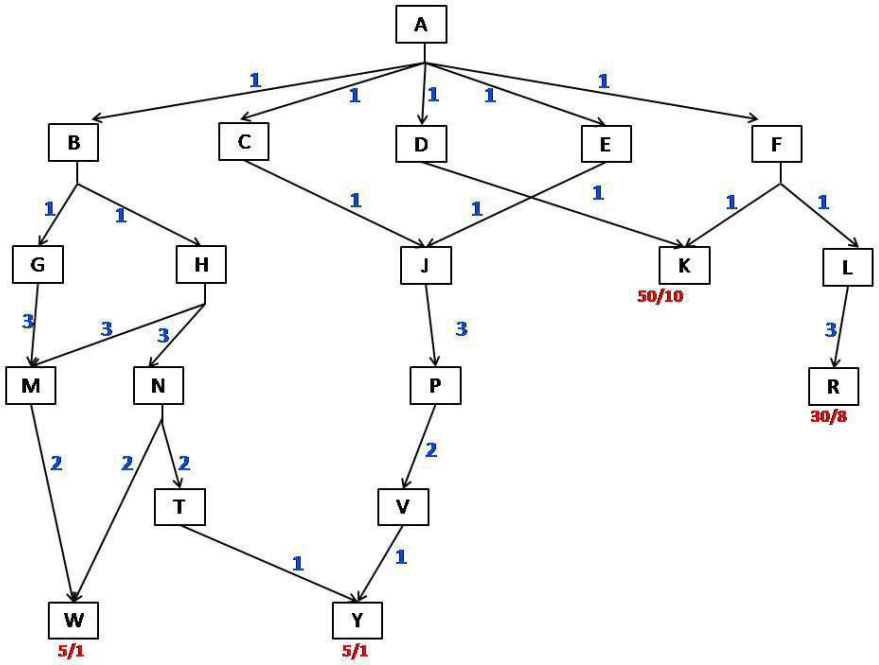

Figure 4.8: The GNG of the minimum cost option of implementation of the graph structure of Fig. 4.7

Example 4 of Fig. 4.11:

- The program examines all the 226,224 G-paths.
- The output was returned after about 7.83 hours.

Example 5 of Fig. 4.12:

- The number of options checked are 350,712.
- The time required was about 19 hours.

Example 6 of Fig. 4.13:

- The number of G-paths that our algorithm examines is 499,518.
- The output was returned after about 38 hours.

Example 7 of Fig. 4.14:

- The program examines all the 1,395,648 options of implementation.
- It required around 13 days of execution before returning the output.

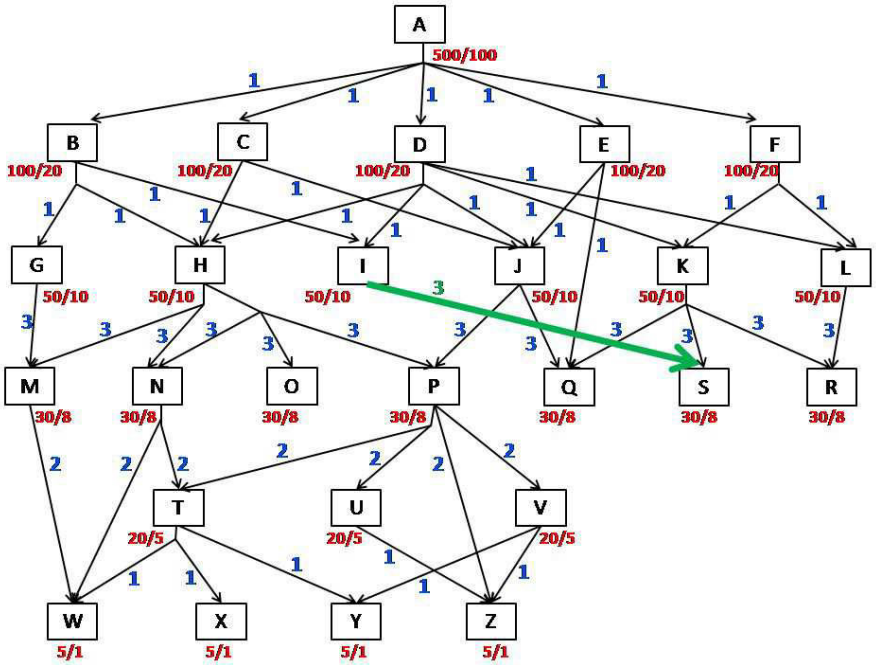

Figure 4.9: Example 2: adding one more hyperarc to the directed hypergraph of Fig. 4.7

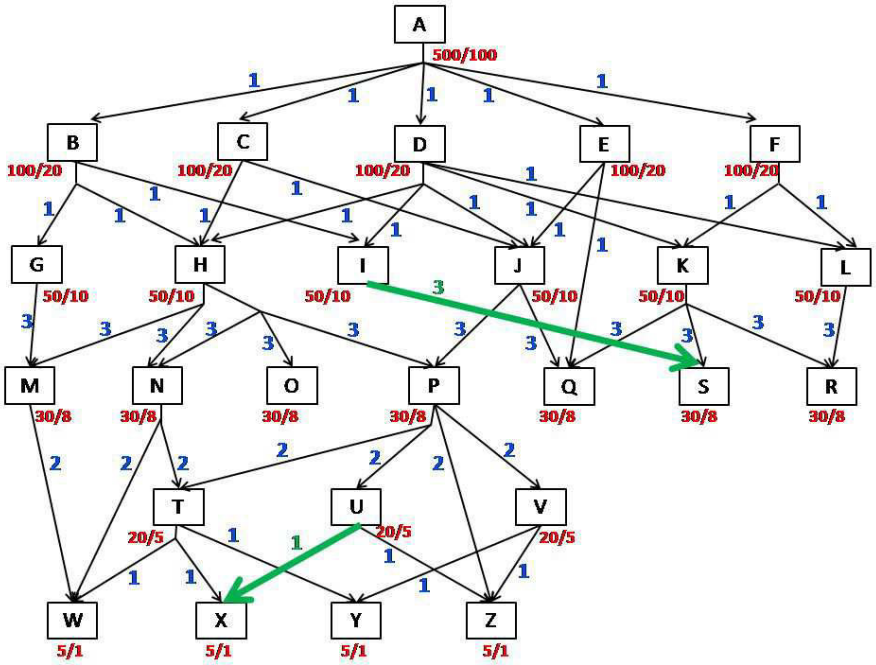

Figure 4.10: Example 3: adding two more hyperarcs to the directed hypergraph of Fig. 4.7

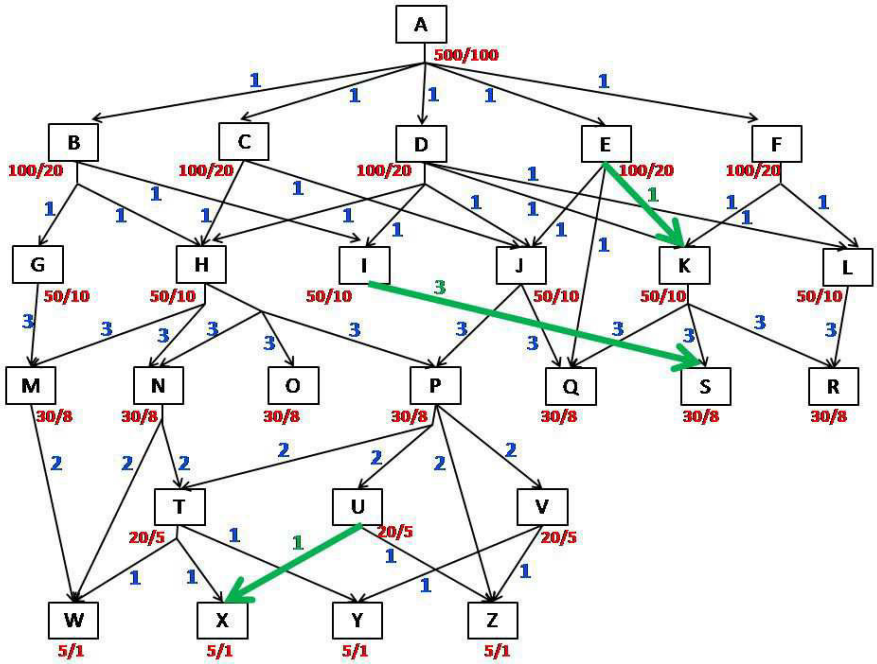

Figure 4.11: Example 4: adding three more hyperarcs to the directed hypergraph of Fig. 4.7

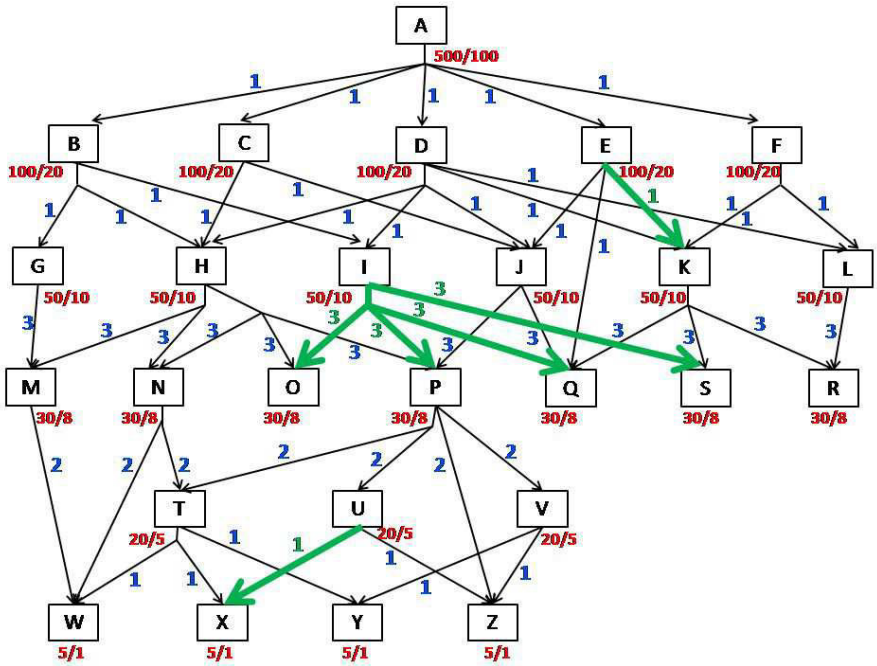

Figure 4.12: Example 5: adding four more hyperarcs to the directed hypergraph of Fig. 4.7

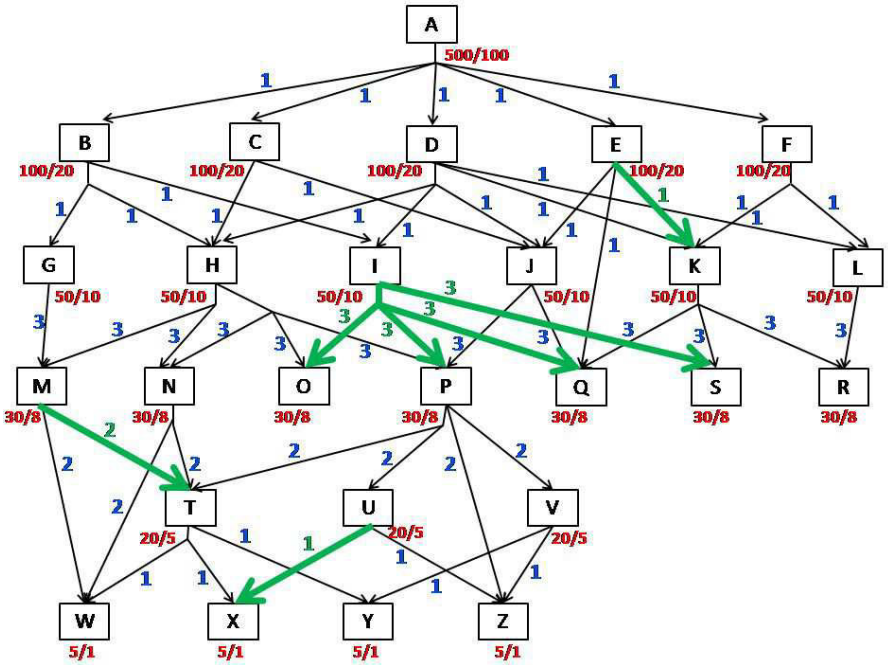

Figure 4.13: Example 6: adding five more hyperarcs to the directed hypergraph of Fig. 4.7

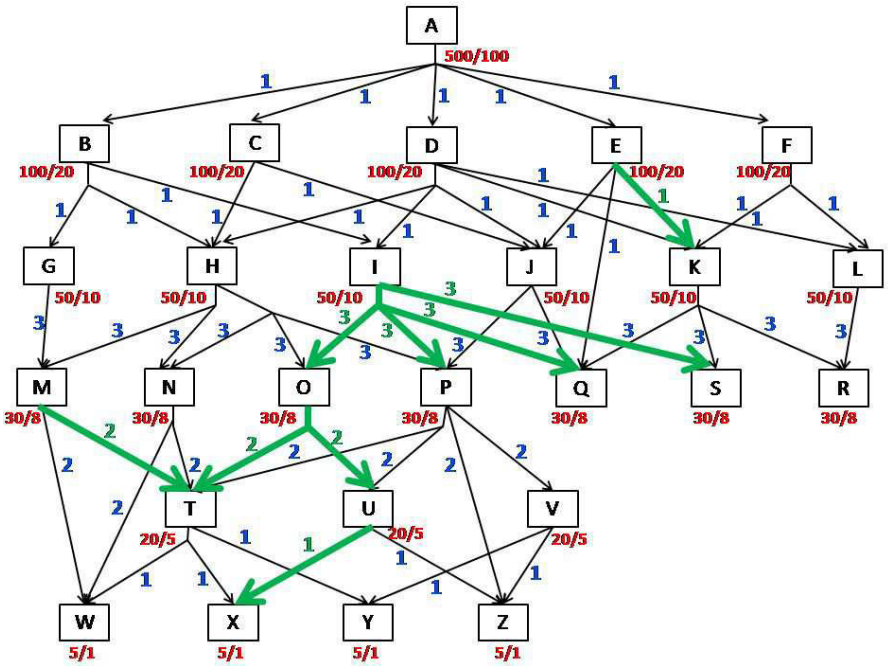

Figure 4.14: Example 7: adding six more hyperarcs to the directed hypergraph of Fig. 4.7

It's obvious that the number of options is growing fast with the increase in the number of hyperarcs, and this growth differs depending on the location of the added hyperarcs. So, it seems clear that our algorithm is a more complex technique than the previously selected SA technique (which doesn't require much time of execution since it's a heuristic) but however, the MCD algorithm is obviously capable of providing an exact-optimal solution unlike the SA technique which yields a near-optimal one. Thus, the designer has a choice of either choosing our more reliable technique which gives the best possible solution but needs more time (but still is less complex than the ES technique since we have ignored some options of implementation in our search), or to accept good solutions which are not necessarily the very best (especially for large space inputs) but require less computing effort.
It's important to note that a long time of execution of an algorithm need not pose a problem in such kind of research, because the algorithm is run only for once before forming the design and afterwards the system will be designed based on the result of the chosen algorithm.

In the example that we exhibit in this section, we evolved the directed hypergraph representation of the multi-standard system by adding hyperarcs to the directed hypergraph. One might ask: why not add nodes or even increase the number of levels in the figure? In fact, adding nodes might most probably yield an increase in the number of hyperarcs of the directed hypergraph, and adding more levels to the figure will certainly increase both its number of nodes and number of hyperarcs. So, one can notice that finally all ends at adding more hyperarcs to the directed hypergraph to make it more complicated, which certainly increases the number of options of implementation for a multi-standard system design.

4.6 Conclusion

Our research dwells with the optimization of the SDR multi-standard system, in order to construct an optimal design, using graph theory. After having proved in the previous chapter that our stated optimization problem is an NP-problem under a certain constraint, we had to find some way to reduce its complexity. For this reason, we have performed in this chapter an exploration on the various types of options of implementation and proved that those whose generated graphs are not G-paths don't contribute for minimum cost designs. Thus, we were able to restrict our study to only the alternatives whose generated graphs are G-paths. This was the first step. Afterwards, we adopted various definitions and notations of directed hypergraphs to exhibit a new algorithm whose role is to identify the G-path of a multi-standard system having the minimum cost. A computational complexity analysis of our algorithm showed that its resources use is exponentially upper bounded and thus, we have presented several examples on which we applied our algorithm in order to give a brief idea on the evolution of its complexity. Although it seemed to require a non-negligible amount of computing time as the directed hypergraph input grows (by adding

more and more hyperarcs for instance) as compared with a near-optimal technique which normally requires negligible computing effort, yet our proposed algorithm gained importance as it provides the best and exact-optimal solution unlike the heuristic tools.

Conclusions and prospects

This thesis has addressed the problem of designing an optimal SDR multi-standard system which balances between flexibility and computing efficiency in the context of the theoretical approach of the parametrization technique, for identifying the most appropriate Common Operators (CO) to be installed in the system supporting several communication standards. Our work was to theoretically model the graphical approach of the multi-standard equipment as well as the optimization problem associated with it using graph theory, and select or develop new theoretical tools to solve it.

General Conclusions

We began this dissertation by describing the evolution of the SoftWare Radio (SWR) technology which was driven by the accelerating rate of standards' emergence, as a replacement of the conventional "Velcro" approach which will become obsolete very soon. This technology, which tends to move the data conversion right next to the antenna and to do all signal processings in the digital domain, is faced with several obstacles related to the RF-front end and the Analog-to-Digital converters which are not able to cope with such high frequencies. To overcome these challenges came the Software-Defined Radio (SDR), the practical version of SWR, which employs Intermediate Frequency (IF) sampling. The parametrization technique was proposed as a methodology to design such SDR systems which consists in selecting the COs, inside and between the different supported standards in the system, whose operations can be modified by a simple parameter adjustment. All these ideas were highlighted in **Chapter 1** but indeed, we have further presented two approaches to identify and develop COs. However, the theoretical approach was explored in greater details as it represents the foundation of this thesis subject.

In the theoretical approach, we discuss the idea of decomposing the supported standards into several layers by a graphical illustration, which provides a pictorial view of all the design alternatives of the multi-standard system. In order to find the best of the available options of implementation, the graphical approach was turned into an optimization problem by stating a cost function which is required to be minimized. There are many cost parameters which can be associated to this problem but among them, we mentioned the BC, CC and NoC which were previously considered to form the suggested objective function. The

BC can represent the number of logic gates, number of execution cycles, area $\cdots$ in an FPGA or a DSP implementation or in a heterogeneous design consisting of both DSPs and FPGAs. CC can be considered in terms of execution time. Based on these cost parameters we presented the cost function using the weighted sum approach, because of its popularity and effectiveness in many multi-objective problems, to aggregate both objectives of minimizing the total BC and total CC. Our objective in this thesis was to optimize this suggested cost function using graph theoretical models and characterizations of different aspects of the problem. This formed our optimization problem, where stochastic techniques which give a near-optimal solution were selected by a previous PhD student to solve it because he was intuitively convinced that it's a hard problem.

In **Chapter 2**, we started with the fundamentals of graph theory by stating various definitions and theorems related to graphs, digraphs, hypergraphs and finally directed hypergraphs. This part provided a clear idea of the different notions and types of graphs, which enabled us later to choose the most adequate modeling variants for our problem. We also presented in this chapter various applications and problems arising in graph theory to highlight some examples of such theoretical modelings. The theory of complexity was further introduced in some details as it was used to study the complexity imposed by our optimization problem. We presented the class of polynomially solved problems, the class P, along with the problems considered to be intractable in NP-P, and finally highlighted the notion of the hardest solvable problems called NP-complete.

The graphical structure of the SDR multi-standard system introduced in chapter one was modeled as a directed hypergraph in **chapter 3**, which was necessary to represent both the "OR" and "AND" dependencies, accompanied with numerical values associated to its different nodes and arcs. This directed hypergraph representation provided all the alternatives capable of implementing the multi-standard system but however, we explained how we can obtain from it our graphical suggestion for any one of these design alternatives, also illustrated as a directed hypergraph and called a generated graph. Moreover, we have formulated in this chapter the cost function presented in chapter one in an alternative equation form as function of the weights of paths in a directed hypergraph using the generated graph of each option. All these theoretical modelings and others have given the problem a formal form stated in a theoretical manner.
Our aim is to select the option of implementation which possesses the least cost imposed by the proposed cost function and thus, we have performed an exploration to find how huge is the number of alternatives from which we are supposed to choose the best one. In fact, we were able to attain an exponential upper bound for this number as function of some considered parameters of the directed hypergraph representation of a multi-standard system. This was a clue that our optimization problem is not an easy one and thus came the idea of proving that it's an NP-problem. For this, we changed it into a decision problem since any optimization problem is at least as hard as its corresponding

decision problem version. Then we presented a detailed proof demonstrating that the associated decision problem to our optimization problem is NP but on condition that the number of levels in the graph structure doesn't exceed a certain constant, which is considered to be usually the case.

After having proved that our optimization problem is complex, we thought of a way to try to reduce the number of options necessary to be tested. In fact we realized that the options of implementation can be divided into those whose generated graphs admit a duplicated part and those illustrated as G-paths. In **Chapter 4** we have proved that when searching for a minimum cost design, we can disregard the alternatives in which duplication occurs in their generated graphs, because we can always find a lower cost design whose generated graph is a G-path. This helped us propose a new Minimum Cost Design (MCD) algorithm to solve our optimization problem, using various modeling notions concerning directed hypergraphs, which examines only the G-path options of implementation instead of testing all of them. The algorithm returns as output the G-path corresponding to a design possessing the minimum cost, from which we can extract the common operators to be installed in the design being the block vertices with out-degree zero. We analyzed the computational complexity of MCD and found out that the resources' requirement is exponentially upper bounded and thus the problem still remains a complex one. In our complexity analysis, we have attained a bound which is worse than the worst case and that's why we considered that we have found an upper bound for the execution time of the algorithm and not the exact one.

Since this was a worst-case analysis, we were convinced that this need not be the required computing effort in most of the cases and so we have applied our algorithm on various generic examples to discover its complexity evolution after having developed a program code for MCD using C language. Results show that MCD does actually require an important computing effort which increases rapidly as we increase the number of hyperarcs in the directed hypergraph representation of a multi-standard system, due to the huge increase in the tested options of implementation. Although our algorithm requires more computing effort than stochastic heuristic techniques which usually need a negligible amount of execution time but however, MCD is capable of providing an exact optimal solution and not a near-optimal one. So, the designer has the choice of either choosing a technique which needs a non-negligible amount of time but provides a more reliable solution, or choosing one which gives a good solution (maybe not the best especially for big size instances) but needs much less time. It's important to highlight the point that time requirements might not constitute a huge restriction behind choosing our technique. Some might for example sacrifice months to get an answer but then design the best optimal multi-standard system.

Prospects

- In this approach common operators will be re-used by several communication blocks, thus imposing scheduling issues at run time. A methodological procedure has to be proposed to arrange the over-use demands of each common block in the design, in which it might be necessary to duplicate certain operators.
- The complexity of the presented optimization problem in this dissertation can be further examined to discover if it is even more complex, i.e by proving it to be an NP-complete problem if possible.
- The cost function was formulated using the weighted sum approach. However, other techniques can be used for such multi-objective optimization problems which can be investigated. In addition, one possibility might be to combine scheduling issues with the current graph optimization problem by adding more objectives to the formulated cost function.
- Our algorithm which gives an exact-optimal solution will be a better tool than the Exhaustive search technique to validate and compare the results of various heuristic tools, if a certain designer insists on using a fast stochastic search technique.
- The exclusion of some of the options of implementation has definitely reduced the complexity of our optimization problem. One question arises which is to explore the amount by which the number of examined options of implementation is reduced, and consequently by how much the complexity of our problem is improved.
- The idea of ignoring the options of implementation whose generated graphs don't correspond to G-paths can be exploited on any other future proposed solution for our problem, whether a deterministic or a heuristic optimization tool, leading to a less complex technique by investigating a less number of alternatives capable of implementing the multi-standard system.

Appendix

Appendix A

The source code

This appendix presents the C program source code for the proposed Minimum Cost Design (MCD) algorithm.

A.1 Structures and Functions for the input

Each node in the figure is represented by an element of type "Node" characterized by its name, BC, CC and the level that it occupies. A linked list structure representing the vertices in the graph is defined as follows:

```
typedef struct node
{
   char name;
   int BC;
   int CC;
   int level;
   struct node *nxt;
}Node;
```

Hyperarcs in the associated directed hypergraph are as well represented by a linked list. Each element in this linked list contains another linked list responsible for creating the unknown number of head nodes in a hyperarc. Both linked lists are introduced hereby. Note that there was no need for a linked list representation of the tail set of the hyperarcs because there is exactly one tail node (the parent node) in each hyperarc.

```
typedef struct HeadNoC
{
   char head;
   int NoC;
   struct HeadNoC *nxt;
}HeadNoC;
```

```
typedef struct Hyperarc
{
   char tail;
   HeadNoC* hNoC;
   struct Hyperarc *nxt;
}Hyperarc;
```

Fig. A.1 shows a sequence of elements each representing one hyperarc in the multi-standard graph structure.

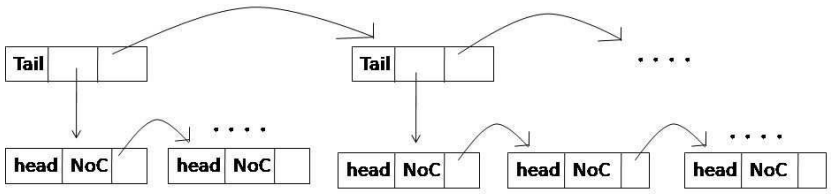

Figure A.1: An illustration of the linked list of type "Hyperarc" creating a set of hyperarcs

The input was created by a structure (called "Graph") that contains the two linked lists, structure "Node" and structure "Hyperarc", as follows:

```
typedef struct graph
{
   Node *N;
   Hyperarc *H;
}Graph;
```

The following three functions develop the three introduced linked list chains. All of them return a pointer which points to the first element of the created linked list.

- *Node* CreationListNode(int NumberOfNodes)*; Creates a chain of elements corresponding to the nodes in the graph. Note that we choose to enter the node r in H_r before any other node in the graph structure of an SDR multi-standard system.
- *HeadNoC* CreationHeadNoC(int NumberOfHeads)*; Develops the list of head nodes of a certain hyperarc.
- *Hyperarc* CreationListHyperarc(int NumberOfHyperarcs)*; Forms a sequence of elements representing the hyperarcs in the input directed hypergraph. The function "*CreationHeadNoC*" is used inside this function.

The details of the code of each of these functions include:

- ```
  Node* CreationListNode(int NumberOfNodes)
  {
     int i;
     Node *FirstNode, *aux, *new;
  ```
  ```

```
    FirstNode=malloc(sizeof(Node));
    printf("Enter the BC, CC, level and name of the first node:
");
    scanf("%d %d %d", &FirstNode->BC, &FirstNode->CC,
&FirstNode->level);
    while(getchar() != '\n');
    FirstNode->name=ReadCharacter();
    aux=FirstNode;
    for(i=1; i<NumberOfNodes; i++)
    {
      new=malloc(sizeof(Nodes));
      printf("Enter the BC, CC, level and name of the %d th node:
",i+1);
      scanf("%d %d %d", &new->BC, &new->CC, &new->level);
      while(getchar() != '\n');
      new->name=ReadCharacter();
      aux->nxt=new;
      aux=aux->nxt;
    }
    aux->nxt=NULL;
    return(FirstNode);
}
```

- ```
  HeadNoC* CreationHeadNoC(int NumberOfHead)
  {
    int i;
    Node *FirstHeadNoC, *aux, *new;
    FirstHeadNoC=malloc(sizeof(HeadNoC));
    printf("Enter the name of the first node in the head set of the
  hyperarc ");
    while(getchar() != '\n');
    FirstHeadNoC->head=ReadCharacter();
    printf("Enter the NoC associated with the first head in the
  head set of the hyperarc ");
    scanf("%d", &FirstHeadNoC->NoC);
    aux=FirstHeadNoC;
    for(i=1; i<NumberOfHead; i++)
    {
      new=malloc(sizeof(HeadNoC));
      printf("Enter the name of the %dth node in the head set of
  the hyperarc: ",i+1);
      while(getchar() != '\n');
      new->head=ReadCharacter();
      printf("Enter the NoC associated with the %dth head in the
  head set of the hyperarc ");
      scanf("%d", &new->NoC);
  ```
  ```

```
      aux->nxt=new;
      aux=aux->nxt;
    }
    aux->nxt=NULL;
    return(FirstHeadNoC);
  }
```

- **Hyperarc* CreationListHyperarc(int NumberOfHyperarcs)**

```
  {
    int i, NumberOfHead;
    Hyperarc *FirstHyperarc, *aux, *new;
    FirstHyperarc=malloc(sizeof(Hyperarc));
    printf("Enter the name of the tail of the first hyperarc: ");
    while(getchar() != '\n');
    FirstHyperarc->tail=ReadCharacter();
    printf("Enter the number of the head nodes of the first hyper-
arc: ");
    scanf("%d", &NumberOfHead);
    FirstHyperarc->hNoC=CreationHeadNoC(NumberOfHead);
    printf("\n \n");
    aux=FirstHeadNoC;
    for(i=1; i<NumberOfHyperarcs; i++)
    {
      new=malloc(sizeof(Hyperarc));
      printf("Enter the name of tail of %dth hyperarc: ",i+1);
      while(getchar() != '\n');
      new->tail=ReadCharacter();
      printf("Enter the number of the head nodes of the %dth
hyperarc: ",i+1);
      scanf("%d", &NumberOfHead);
      new->hNoC=CreationHeadNoC(NumberOfHead);
      printf("\n \n");
      aux->nxt=new;
      aux=aux->nxt;
    }
    aux->nxt=NULL;
    return(FirstHyperarc);
  }
```

Note that characters were entered in a different way than integers since the enter '\n' is considered as a character. To avoid this, we were in need of the following function called "ReadCharacter" to input characters (instead of using the *scanf* command) preceded by the code "while (getchar() != '\n');":

```
char ReadCharacter();
{
```

```
    char character;
    character=getchar();
    character=toupper(character);
    while(getchar() != '\n');
    return character;
}
```

A.2 Remaining Structures and Functions for our program code

A.2.1 Remaining structures

- Any option of implementation generated by the algorithm is of the form of a G-path. In our program code, it is necessary to provide a logical representation of an option of implementation. Our choice was to consider that every G-path is sufficiently characterized by the set of hyperarcs that it occupies. The following is a linked list structure, called "ListOptions", corresponding to a set of G-paths:

  ```
  typedef struct ListOptions
  {
     Hyperarc* A;
     struct ListOptions *nxt;
  }ListOptions;
  ```

  It contains only one element, which is a pointer (called "A") to the first element of a linked list of type "Hyperarc", which corresponds to the group of hyperarcs that the option possesses. As an example, fig. A.2 illustrates a sequence of elements of type "ListOptions" each representing one option of implementation.

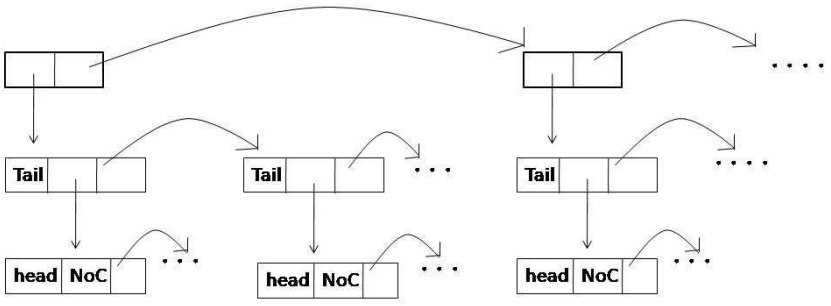

Figure A.2: An illustration of the linked list of type "ListOptions" creating a set of options of implementation

- This linked list structure, called "LetterOnly", reserves the character names of certain desired nodes in the figure and is defined as follows:

```
typedef struct letteronly
{
   char chr;
   struct letteronly* nxt;
}LetterOnly;
```

- In our algorithm, for every selected option of implementation "X", we have introduced a vector "k_v" on every vertex v of X, which is exploited to calculate the cost of the option "X".
  The vector "k_v" will be represented in the code by a linked list called "Vector" which only contains an integer variable called "val". Each element of this list will correspond to an entry in the vector "k_v". This linked list structure is defined as:

```
typedef struct vector
{
   int val;
   struct vector *nxt;
}Vector;
```

- Each vertex in the set Q of the algorithm, in which all the vertices in the selected G-path "X" will be invoked step by step, is represented by an element in the linked list called "Letter", defined as follows:

```
typedef struct letter
{
   char chr;
   int BC;
   int CC;
   int level;
   Vector *V;
   struct letter *nxt;
}Letter;
```

  For each node v which will be invoked in the set Q, we will return to the **input** linked list of nodes (of type *"Nodes"*) and get all the information about this node including its character name, BC, CC, & level. Still, we need to create the vector "k_v" and thus we include a pointer to the first element of the recently introduced linked list of type "Vector", corresponding to the vector "k_v".

- During the calculation process of the cost, we introduced in the algorithm a set A in which we reserve the nodes which have out-degree zero in the option "X" in hand, in order to finally add their BCs to the cost S (according to step 8 of the algorithm). Thanks to the linked list structure, as we will once again use it to create the last structure in our program

corresponding to the set A in the algorithm. Each element in this list will include a character representing the name of the node along with the only necessary information in this case which will be the BC of the vertex. This linked list structure (called "LetterBC") will be defined by the following:

```
typedef struct letterBC
{
   char chr;
   int BC;
   struct letterBC *nxt;
}LetterBC;
```

A.2.2 Remaining functions

1. *ListOptions * CopyOption(ListOptions *X)*; Creates a copy of a certain selected G-path X. It needs only one input parameter, which is a pointer to the option that needs to be copied. It returns a variable of type "ListOption *", corresponding to a pointer to the newly created copy of the option. Code details of this function are presented hereby.

```
ListOptions * CopyOption(ListOptions *X)
{
   ListOptions *perm=malloc(sizeof(ListOptions));                1
   perm->A=malloc(sizeof(Hyperarc));                             2
   Hyperarc *perm1=perm->A;                                      3
   Hyperarc *tmp1=X->A;                                          4
   perm1->tail=tmp1->tail;                                       5
   perm1->hNoC=malloc(sizeof(HeadNoC));                          6
   HeadNoC *perm2=perm1->hNoC;                                   7
   HeadNoC *tmp2=tmp1->hNoC;                                     8
   perm2->head=tmp2->head;                                       9
   perm2->NoC=tmp2->NoC;                                        10
   tmp2=tmp2->nxt;                                              11
   while(tmp2 != NULL)                                          12
  {
      HeadNoC *new=malloc(sizeof(HeadNoC));                     13
      new->head=tmp2->head;                                     14
      new->NoC=tmp2->NoC;                                       15
      tmp2=tmp2->nxt;                                           16
      perm2->nxt=new;                                           17
      perm2=new;                                                18
  }
   perm2->nxt=NULL;                                             19
   tmp1=tmp1->nxt;                                              20
   while(tmp1 != NULL)                                          21
  {
```

```
      Hyperarc *new1=malloc(sizeof(Hyperarc));                    22
      new1->tail=tmp1->tail;                                      23
      new1->hNoC=malloc(sizeof(HeadNoC));                         24
      HeadNoC *perm2=new1->hNoC;                                  25
      HeadNoC *tmp2=tmp1->hNoC;                                   26
      perm2->head=tmp2->head;                                     27
      perm2->NoC=tmp2->NoC;                                       28
      tmp2=tmp2->nxt;                                             29
      while(tmp2 != NULL)                                         30
     {
        HeadNoC *new=malloc(sizeof(HeadNoC));                     31
        new->head=tmp2->head;                                     32
        new->NoC=tmp2->NoC;                                       33
        tmp2=tmp2->nxt;                                           34
        perm2->nxt=new;                                           35
        perm2=new;                                                36
     }
      perm2->nxt=NULL;                                            37
      tmp1=tmp1->nxt;                                             38
      perm1->nxt=new1;                                            39
      perm1=new1;                                                 40
   }
    perm1->nxt=NULL;                                              41
    perm->nxt=NULL;                                               42
    return(perm);                                                 43
}
```

Lines 1 till 42 of the code perform a copy of the option pointed by X and in line 43, we return this copy.

2. *void AddHyperarcToOption(ListOptions *Option, Hyperarc *perm)*; adds a certain selected hyperarc to the group of hyperarcs already existing in a specific option of implementation. This function needs two input parameters; one is a pointer to an element of type "ListOptions" representing the associated option of implementation, & the other is a pointer to an element of type "Hyperarc" corresponding to the hyperarc that will be copied. This function's code details involve:

```
void AddHyperarcToOption(ListOptions *Option, Hyperarc *perm)
{
   Hyperarc *copy=malloc(sizeof(Hyperarc));                       1
   copy->tail=perm->tail;                                         2
   copy->hNoC=malloc(sizeof(HeadNoC));                            3
   HeadNoC *copy1=copy->hNoC;                                     4
   HeadNoC *perm1=perm->hNoC;                                     5
   copy1->head=perm1->head;                                       6
   copy1->NoC=perm1->NoC;                                         7
   perm1=perm1->nxt;                                              8
   while(perm1 != NULL)                                           9
  {
```

```
        HeadNoC *new=malloc(sizeof(HeadNoC));                    10
        new->head=perm1->head;                                   11
        new->NoC=perm1->NoC;                                     12
        perm1=perm1->nxt;                                        13
        copy1->nxt=new;                                          14
        copy1=new;                                               15
    }
     copy1->nxt=NULL;                                            16
     copy->nxt=Option->A;                                        17
     Option->A=copy;                                             18
}
```

Lines 1 till 16 of the code perform a copy of the hyperarc pointed by "perm". This new copy will be added to the beginning of the linked list of hyperarcs present in the option pointed by *"Option"*, which is done in lines 17 and 18.
This function, along with the "CopyOption" function, will be able to form the generated G-path "$X + E$" of the algorithm.

3. *int CheckIfNodeHasEmptyForwardStar(ListOptions *Y, char vertex)*; its role is to check whether a specific selected vertex has out-degree zero (in which case the function returns "1") or not (then the function returns "**0**") in a certain selected G-path. It will be used for step 12 of the algorithm as well as to check the condition of step 7. It needs two input parameters; the first parameter being a pointer to an element of type "ListOptions" which stands for a certain option of implementation, while the second parameter will be a character that represents the name of the node that is going to be checked for emptiness of its forward star in the corresponding G-path. Details of its code statements include:

```
int CheckIfNodeHasEmptyForwardStar(ListOptions *Y, char ver-
tex)
{
    int a=1;                                                     1
    Hyperarc *U=Y->A;                                            2
    while((U != NULL) && (a == 1))                               3
   {
      if(U->tail == vertex)                                      4
     {
      a=0;                                                       5
     }
      U=U->nxt;                                                  6
   }
    return(a);                                                   7
}
```

In fact, in the statements of this function, we verify if the character node *"vertex"* is a tail node of any of the hyperarcs present in the option of imple-

mentation pointed by Y (see condition of line 4). If so, then "vertex" clearly doesn't have an empty forward star in this option and we shall return "**0**" as an integer (line 5). But if after having examined all the hyperarcs we found out that "vertex" isn't a tail node of any, then we would conclude that its forward star in this option is empty and thus return the integer "**1**".

4. Deletion of an option of implementation: Our aim is to accomplish the "**remove** $X \in M$" part of the statement "**select and remove** $X \in M$" of the algorithm. However, we will need to invoke one function inside another. Accordingly, we will use the function *EraseElements* inside the *DeleteOption* function.
   - *void EraseElement(ListOptions *X)*; erases the elements of the option (using the library function *free*), element by element, from the elements of type *HeadNoC* up to those of type *Hyperarc* and finally up to the element of type *ListOption* pointed by X. It only needs as input parameter a pointer to the element of type "ListOptions" which we desire to delete. It includes the following code statements:

```
void EraseElement(ListOptions *X)
{
   Hyperarc *Q1=X->A;
   Hyperarc *Q2=X->A;
   while(Q1 != NULL)
            1
            2
            3
  {
     HeadNoC *P1=Q1->hNoC;
     HeadNoC *P2=Q1->hNoC;
     while(P1 != NULL)
            4
            5
            6
    {
       P2=P2->nxt;
       free(P1);
       P1=P2;
            7
            8
            9
    }
     Q2=Q2->nxt;
     free(Q1);
     Q1=Q2;
          10
          11
          12
  }
   free(X);
```

```
            13
}
```

- *void DeleteOption(ListOptions **FirstOption, ListOption *X);* tests if the option that we want to remove is the first element in the linked list of options (of type "ListOptions") or not, where different deleting strategies will be used in either cases. It requires two input parameters; one is a pointer to the element from the linked list of type "ListOptions" which we want to delete. The other parameter is of type "ListOptions **" which corresponds to the first element of this linked list. Remark that the **passage by address** strategy is used for this parameter by referring to the address of a pointer (pointer to a pointer) instead of the pointer itself, which is required since the first element in the linked list of options might change (which is the case when it deletes the first element of the chain). Hereby are the code details of this function:

```
void DeleteOption(ListOption **FirstOption, ListOption *X)
{
   if(X == *FirstOption)
            1
  {
     *FirstOption=(*FirstOption)->nxt;
     EraseElement(X);
            2
            3
  }
   else
            4
  {
     ListOptions *tmp=*FirstOption;
     while(tmp->nxt != X)
            5
            6
    {
       tmp=tmp->nxt;
            7
    }
     tmp->nxt=X->nxt;
     EraseElement(X);
            8
            9
  }
}
```

In this function, if the option we desire to delete was the first one, we advance the pointer to the next element before erasing it (lines 1 till 3). Otherwise, we search for the element just preceding that pointed

by X and link it to the element following that at which X points (lines 4 till 9), and finally erase it.

5. *int NotRepetition(LetterOnly **L, char name)*; This function was developed because in the code, we face situations where we need to test all the head nodes of all the hyperarcs in a certain option of implementation. This possibly imposes multiple tests for the same vertex. The *"NotRepetition"* function checks if a specific node has already been selected or not in order to avoid repeating certain required tasks. In fact, after having reserved the character names of all the selected vertices so far in a linked list of type *LetterOnly*, the *"NotRepetition"* function checks if the character name of the node in hand is an element in this list. If it wasn't, it adds it to this linked list indicating that this vertex has been selected and there will be no need to test it again if it was selected later on (see lines 8 till 17 of the code). This function returns an integer value, which is either "**0**" or "**1**" respectively depending on whether the node we're testing has already been selected before or not. It accepts two input parameters; the first is a **pointer to a pointer** of type "LetterOnly **", which will correspond to the first element in a linked list of type "LetterOnly". The second input parameter for this function is of type "char" and represents the name of the node which is desired to check for repeatidness. The statements of this function include:

```
int NotRepetition(LetterOnly **L, char name);
{
   int a=1;                                                    1
   LetterOnly *P1=*L, *P2=*L;                                  2
   while((P1 != NULL) && (a == 1))                             3
  {
     if(P1->chr == name)                                       4
    {
       a=0;                                                    5
    }
     P2=P1;                                                    6
     P1=P1->nxt;                                               7
  }
   if(a == 1)                                                  8
  {
     if(*L == NULL)                                            9
    {
       *L=malloc(sizeof(LetterOnly));                          10
       (*L)->chr=name;                                         11
       (*L)->nxt=NULL;                                         12
    }
     else                                                      13
    {
```

```
            LetterOnly *new=malloc(sizeof(LetterOnly));          14
            new->chr=name;                                       15
            P2->nxt=new;                                         16
            new->nxt=NULL;                                       17
        }
    }
    return(a);                                                   18
}
```

6. Deletion of a vertex v from the set Q: in the MCD algorithm, we will need at some point to eliminate a selected element v from Q (the **"select & remove** $v \in Q$**"** statement). This will be done using the function *"DeleteVertex"* which calls another function called *"EraseElementVertex"*. The two functions resemble to the previously explained "DeleteOption" and "EraseElement" functions respectively, so we will present right next their code statements without providing any explanation.

```
void EraseElementVertex(Letter *selectedQ)
{
    Vector *Q1=selectedQ->V;
    Vector *Q2=selectedQ->V;
    while(Q1 != NULL)
    {
      Q2=Q2->nxt;
      free(Q1);
      Q1=Q2;
    }
    free(selectedQ);
}

void DeleteVertex(Letter **Q, Letter *selectedQ)
{
    if(selectedQ == *Q)
    {
      *Q=(*Q)->nxt;
      EraseElementVertex(selectedQ);
    }
    else
    {
      Letter *tmp1=*Q;
      while(tmp1->nxt != selectedQ)
      {
          tmp1=tmp1->nxt;
      }
      tmp1->nxt=selectedQ->nxt;
      EraseElementVertex(selectedQ);
    }
```

}

7. *void WorkInCost(ListOptions *Option, Letter *selectedQ, Letter *Q, Graph G,*
*int *D, LetterBC ** A)*: performs the tasks of steps 3 till 7 of the algorithm. It needs six input parameters which include:
   - a pointer of type "ListOptions *" pointing to the selected option of implementation "X".
   - a pointer of type "Letter *" called "selectedQ", pointing to the element associated to the selected vertex v in Q.
   - another pointer of type "Letter *" called "Q", pointing to the first element in the linked list of type "Letter" (where this linked list corresponds to the vertices present in the set Q).
   - a variable of type "Graph", representing the input linked lists of nodes and hyperarcs.
   - a pointer to an integer value (of type int *) which stands for the variable S in which the cost of the selected G-path will be accumulated.
   - a **pointer to a pointer** of the first element in the linked list of type "LetterBC" (where this linked list corresponds to the vertices in the set A).

   Note that in the last two parameters, we use a pointer to the desired values, where the desired values include an integer corresponding to the cost S & a pointer to the first element in the linked list of type "LetterBC". This is because if we send the desired values by themselves into the function, then a copy of these values will be realized which will be different than the ones in the main code. These copy values will be destroyed at the end of the function and will be deleted. While, by using a pointer to them, the desired values will be preserved after leaving the function and this is what we actually needed.
   The code statements of the **WorkInCost** function are presented as follows:

```
void WorkInCost(ListOptions *Option, Letter *selectedQ,
                Letter *Q, Graph G, int *D, LetterBC ** A)
{
   Hyperarc *prm1=Option->A;                                    1
   while(prm1 != NULL)                                          2
  {
     if(prm1->tail == selectedQ->chr)                           3
    {
       HeadNoC *prm2=prm1->hNoC;                                4
       while(prm2 != NULL)                                      5
      {
         int a=1;                                               6
         Letter *Q2=Q, *Q3=Q;                                   7
         while((Q2 != NULL) && (a == 1))                        8
        {
```

```
    if(Q2->chr == prm2->head)                                    9
    {
       a=0;                                                      10
    }
    Q3=Q2;                                                       11
    Q2=Q2->nxt;                                                  12
}
 if(a == 1)                                                      13
{
    Node *Gn=G.N;                                                14
    while(Gn->name != prm2->head)                                15
    {
       Gn=Gn->nxt;                                               16
    }
    Letter *new=malloc(sizeof(Letter));                          17
    new->chr=Gn->name;                                           18
    new->BC=Gn->BC;                                              19
    new->CC=Gn->CC;                                              20
    new->level=Gn->level;                                        21
    new->V=NULL;                                                 22
    Q3->nxt=new;                                                 23
    new->nxt=NULL;                                               24
    Q3=new;                                                      25
}
 Vector *ax1, *ax2;                                              26
 if(a=0)                                                         27
{
    ax2=Q3->V;                                                   28
    while(ax2->nxt != NULL)                                      29
    {
       ax2=ax2->nxt;                                             30
    }
    ax1=ax2;                                                     31
}
 Vector *variable=selectedQ->V;                                  32
 while(variable != NULL)                                         33
{
    if(Q3->V == NULL)                                            34
    {
       Q3->V=malloc(sizeof(Vector));                             35
       Q3->V->val=(variable->val)*
                                                                 36
                  (prm2->NoC);
       Q3->V->nxt=NULL;                                          37
       ax1=Q3->V, ax2=Q3->V;                                     38
    }
    else                                                         39
    {
       Vector *new1=malloc(sizeof(Vector));                      40
```

```
        new1->val=(variable->val)*
                    (prm2->NoC);                                    41
        ax1->nxt=new1;                                              42
        new1->nxt=NULL;                                             43
        ax1=new1;                                                   44
      }
      variable=variable->nxt;                                       45
    }
    if(CheckIfNodeHasEmptyForwardStar(Option,
                                      prm2->head))                  46
    {
      if(a == 0)                                                    47
      {
        ax2=ax2->nxt;                                               48
      }
      while(ax2 != NULL)                                            49
      {
        *D=*D+(ax2->val)*(Q3->CC);                                  50
        ax2=ax2->nxt;                                               51
      }
      int b=1;                                                      52
      LetterBC *Y=*A, *Z=*A;                                        53
      while((Y != NULL) && (b == 1))                                54
      {
        if(Y->chr == prm2->head)                                    55
        {
          b=0;                                                      56
        }
        Z=Y;                                                        57
        Y=Y->nxt;                                                   58
      }
      if(b == 1)                                                    59
      {
        if ((*A) == NULL)                                           60
        {
          (*A)=malloc(sizeof(LetterBC));                            61
          (*A)->chr=Q3->chr;                                        62
          (*A)->BC=Q3->BC;                                          63
          (*A)->nxt=NULL;                                           64
        }
        else                                                        65
        {
          LetterBC *new2=
              malloc(sizeof(LetterBC));                             66
          new2->chr=Q3->chr;                                        67
          new2->BC=Q3->BC;                                          68
          Z->nxt=new2;                                              69
          new2->nxt=NULL;                                           70
        }
```

```
            }
             DeleteVertex(&Q, Q3);                                    71
          }
           prm2=prm2->nxt;                                            72
        }
      }
       prm1=prm1->nxt;                                              73
    }
}
```

Consider the following notes:

- In this function, "$Q3$" is made to point to the element of type "Letter" associated to the "h" vertex in the algorithm, while "selectedQ" points to that associated to the selected vertex "v" from the set Q. "$Q3{->}V$" points to the first element in the linked list corresponding to the k_h vector, while "selectedQ$->V$" points to the first element in the linked list corresponding to the k_v vector.
- "prm1" points to the hyperarc in the G-path pointed by "Option" whose tail node is "v", and "h" is one of this hyperarcs' head nodes which is pointed by "prm2".
- The statement $Q = Q \cup \{h\}$ in the algorithm is performed in lines 6 through 25 in this function. Note that if the head node "h" pointed by "prm2" was already an element in the linked list representing the set Q, then the variable a becomes 0. Otherwise, a remains equal to 1 and "h" will be added to the list.
- Lines 27 till 31 execute step 4 in the algorithm of searching for the last filled element in the list of k_h. This step is accessed only if $a = 0$ because otherwise, "$Q3{->}V$" will be made to point to "NULL" (as indicated in line 22 of the code), and thus will have to start filling entries starting from the beginning (as in fact there are no elements yet).
- Steps 5 & 6 of the algorithm are accomplished in lines 32 till 45. Note that "variable->val" corresponds to an entry in "k_v" and "prm2->NoC" stands for the weight on the (v, h) arc.
- The tasks of step 7 of the algorithm are accomplished in lines 46 through 71. This step is accessed only if the vertex h associated to the head node pointed by "prm2" has out-degree zero in the G-path "Option" in hand, which condition is verified in line 46. Note that the variable "ax2->val" corresponds to a newly created entry in "k_h" and "$Q3{->}$CC" stands for the CC of the "h" vertex associated to that pointed by "$Q3$". The final task in step 7 of the algorithm will be to add the vertex "h" to the set A, in case it doesn't already exist. This is performed in lines 52 till 70.
- In order to save a little bit in the execution, we choose to delete the "h" vertex from the set Q directly inside the function (in line 71) and

not outside in the main code after we select it from Q. This is because in this case, the vertex "h" will have an empty forward star in this option (pointed by "Option") and thus the first statement of step 3 of the algorithm won't be satisfied.

8. *void DisplayOption(ListOptions *K)*; Displays an option of implementation as an output. It only requires one input parameter which is a pointer to the option to be displayed. This function's code statements are as follows:

```
void DisplayOption(ListOptions *K)
{
   int i=1;
   Hyperarc *tmp=K->A;
   while(tmp != NULL)
   {
     printf(" Tail of %dth hyperarc in this option is %c \n: ",i,
tmp->tail);
     HeadNoC *perm=tmp->hNoC;
     while(perm != NULL)
     {
       printf("%c ",perm->head);
       perm=perm->nxt;
     }
     printf("\n \n");
     tmp=tmp->nxt;
     i++;
   }
}
```

In this function, we access each hyperarc in the option and print its tail, then we print all its head nodes by accessing the linked list of head nodes of this hyperarc. Once a list of the tail and head nodes of the hyperarcs in the minimum cost option is displayed (using the "DisplayOption" function), one can plot the G-path whose set of hyperarcs includes these ones. This will enable to identify the common operators to install in the design, which will be the blocks with out-degree zero in this illustrated G-path.

9. int NotRepetitionOption(ListOptions *AllOptions, ListOptions *Y); Checks if a generated option pointed by "Y" is one of the previously generated options whose first element in the associated linked list is pointed by the pointer called "AllOptions". If so, it returns the integer "0"; otherwise, this function returns "1".
The code statements of the **NotRepetitionOption** function include:

```
int NotRepetitionOption(ListOptions *AllOptions, ListOptions *
Y)
{
```

```
Hyperarc *U=Y->A;                                                    1
ListOptions *O=AllOptions;                                           2
int c=1;                                                             3
while((O != NULL) && (c == 1))                                       4
{
   Hyperarc *W = O->A;                                               5
   int b=1;                                                          6
   while((W != NULL) && (b == 1))                                    7
  {
     if(U->tail == W->tail);                                         8
    {
       b=0;                                                          9
       HeadNoC *U1=U->hNoC, *W1=W->hNoC;                            10
       int a=1;                                                     11
       while((U1 != NULL) && (a == 1))                              12
      {
         if(U1->head == W1->head)                                   13
        {
           U1=U1->nxt;                                              14
           W1=W1->nxt;                                              15
        }
         else                                                       16
        {
           a=0;                                                     17
        }
      }
       if(a == 1)                                                   18
      {
         U=U->nxt;                                                  19
      }
       else                                                         20
      {
         if(O->nxt != NULL)                                         21
        {
           O=O->nxt;                                                22
           U=Y->A;                                                  23
        }
      }
    }
     else                                                           24
    {
       W=W->nxt;                                                    25
    }
  }
   if(b == 1);                                                      26
  {
     if(O->nxt != NULL)                                             27
    {
```

```
            O=O->nxt;                                          28
            U=Y->A;                                            29
          }
           else                                                30
          {
           O=O->nxt;                                          31
          }

        }
         if(U == NULL)                                         32
        {
           Hyperarc *R=Y->A, *S=O->A;                          33
           int i=0, j=0;                                       34
           while(R != NULL)                                    35
          {
            i++;                                               36
            R=R->nxt;                                          37
          }
           while(S != NULL)                                    38
          {
            j++;                                               39
            S=S->nxt;                                          40
          }
           if(i == j)                                          41
          {
            c=0;                                               42
            EraseElement(Y);                                   43
          }
           else                                                44
          {
            if(O->nxt != NULL)                                 45
            {
              O=O->nxt;                                        46
              U=Y->A;                                          47
            }
          }
        }
      }
     return(c);                                                48
}
```

The hyperarcs in the option pointed by "Y" are accessed using the pointer "U". We check if the hyperarc pointed by "U" is also a hyperarc in the option pointed by "O". For this, we access to the head nodes of all the hyperarcs in "O" whose tail nodes are the same tail node of the hyperarc pointed by "U" (if any), to verify if one of them is the same hyperarc. This is done in lines 4 till 31.
Lines 32 to 47 are accessed only if all the hyperarcs in the option pointed

by "Y" are found in the option currently pointed by "O". In this case, we verify if they do possess the same number of hyperarcs to avoid the possibility that the option pointed by "O" contains more than that pointed by "Y" and thus will correspond to different options.

A.3 The main code

```
void main()
{
   int nbn, nbh, NumberOfLevels, SMin;                                    1
   int NumberOfOptions=1;                                                 2
   ListOptions *K;                                                        3
   Graph G;                                                               4
   printf("enter the number of nodes, number of hyperarcs,                5
          and number of levels in the graph: ");                          6
   scanf("%d %d %d" &nbn, &nbh, &NumberOfLevels);                         7
   G.N=CreationListNode(nbn);                                             8
   G.H=CreationListHyperarc(nbh);

   ListOptions *FirstOption=malloc(sizeof(ListOptions));                  9
   FirstOption->A=malloc(sizeof(Hyperarc));                              10
   FirstOption->A->tail=G.H->tail;                                       11
   FirstOption->A->hNoC=malloc(sizeof(HeadNoC));                         12
   HeadNoC *H1=G.H->hNoC;                                                13
   HeadNoC *H2=FirstOption->A->hNoC;                                     14
   H2->head=H1->head;                                                    15
   H2->NoC=H1->NoC;                                                      16
   H1=H1->nxt;                                                           17
   while(H1 != NULL)                                                     18
  {
     HeadNoC *new=malloc(sizeof(HeadNoC));                               19
     new->head=H1->head;                                                 20
     new->NoC=H1->NoC;                                                   21
     H1=H1->nxt;                                                         22
     H2->nxt=new;                                                        23
     H2=new;                                                             24
  }
   H2->nxt=NULL;                                                         25
   FirstOption->A->nxt=NULL;                                             26
   FirstOption->nxt=NULL;                                                27

   ListOptions *H;                                                       28
   H=CopyOption(FirstOption);                                            29
   H->nxt=NULL;                                                          30

   while(FirstOption != NULL)                                            31
```

```
{
   int S = 0;                                                    32
   LetterBC *A=NULL;                                             33
   Letter *Q=malloc(sizeof(Letter));                             34
   Q->chr=G.N->name;                                             35
   Q->BC=G.N->BC;                                                36
   Q->CC=G.N->CC;                                                37
   Q->level=G.N->level;                                          38
   Q->V=malloc(sizeof(Vector));                                  39
   Q->V->val=1;                                                  40
   Q->V->nxt=NULL;                                               41
   Q->nxt=NULL;                                                  42
   while(Q != NULL)                                              43
  {
     Letter *Q1=Q;                                               44
     while(Q1 != NULL)                                           45
    {
       if(Q1->level == NumberOfLevels)                           46
      {
         Letter *selectedQ=Q1;                                   47
         WorkInCost(FirstOption, selectedQ, Q,
                                          G, &S, &A);            48
         Q1=Q1->nxt;                                             49
         DeleteVertex(&Q, selectedQ);                            50
      }
       else                                                      51
      {
         Q1=Q1->nxt;                                             52
      }
    }
     NumberOfLevels--;                                           53
  }
   while(A != NULL)                                              54
  {
     S=S+(A->BC);                                                55
     A=A->nxt;                                                   56
  }
   if(FirstOption->A->nxt == NULL)                               57
  {
     SMin=S;                                                     58
     K=CopyOption(FirstOption);                                  59
  }
   else                                                          60
  {
     if(S < SMin)                                                61
    {
       SMin=S;                                                   62
       K=CopyOption(FirstOption);                                63
```

```
        }
      }
      ListOptions *X=FirstOption;                                    64
      Hyperarc *T=X->A;                                              65
      LetterOnly *L=NULL;                                            66
      while(T != NULL)                                               67
      {
        HeadNoC *tmp=T->hNoC;                                        68
        while(tmp != NULL)                                           69
        {
          if((CheckIfNodeHasEmptyForwardStar
            (X,tmp->head)) &&                                        70
            (NotRepetition(&L,tmp->head)))
          {
            Hyperarc *perm=G.H;                                      71
            while(perm != NULL)                                      72
            {
              if(perm->tail == tmp->head)                            73
              {
                int d;                                               74
                ListOptions *Y;                                      75
                Y=CopyOption(X);                                     76
                AddHyperarcToOption(Y,perm);                         77
                d=NotRepetitionOption(H,Y);                          78
                if(d == 1);                                          79
                {
                  Y->nxt=FirstOption;                                80
                  FirstOption=Y;                                     81
                  NumberOfOptions++;                                 82
                  ListOptions *Y1;                                   83
                  Y1=CopyOption(Y);                                  84
                  Y1->nxt=H;                                         85
                  H=Y1;                                              86
                }
              }
              perm=perm-> nxt;                                       87
            }
          }
          tmp=tmp->nxt;                                              88
        }
        T=T->nxt;                                                    89
      }
      DeleteOption(&FirstOption,X);                                  90
    }
```

```
    DisplayOption(K);
    printf("The cost of the minimum cost design is %d \n",          91
           SMin);                                                   92
    printf("The number of options tested is %d",                    93
           NumberOfOptions);
}
```

We will highlight some points of this code.

- Lines 1 till 8 initialize some variables and are statements to help enter the input.
- As an initialization in *the algorithm*, we start with the G-path consisting of only the imaginary top vertex r and no hyperarcs ($M = \{(\{r\}, \phi)\}$). This will be the first option that will be tested, from which all the remaining options will be generated. However, this option has no technical significance, as selecting the imaginary vertex r is not an option of implementation. The option $(\{r\}, \phi)$ generates one option, which is $(\{r\} \cup \Im, E_r)$ (call it X_1), due to the way by which the directed hypergraph H_r is defined. The option X_1 corresponds to a logical option of implementation which is to install the top level standards in the design (corresponding to the "Velcro" approach). In our *program code*, we start with the logical X_1 option to stand for an initialization of the options of implementation, instead of starting from the imaginary option $(\{r\}, \phi)$. Lines 9 through 27 perform this initialization, where the option corresponding to X_1 will be pointed by "FirstOption". Note that "FirstOption" will point to the first element in the linked list of options of implementation of type "ListOptions" all along the code.
- The initialization in the statement $Q = \{r\}$ of the algorithm is performed in lines 32 till 42 of the main code. Recall that since we agreed to enter the node r before any other node in the graph structure of an SDR multi-standard system, then $G.N$ (which will be a pointer to the first entered element of type "Node"), will be pointing to the element corresponding to the node r.
- Lines 43 through 53 do the tasks of steps 3 till 7 of the algorithm along with the statement "**select and remove** $v \in Q$". Recall that among these lines, we choose the vertices from Q from those of highest levels in Q till those of lowest levels as required.
- Lines 31 and 43 assure the conditions "until $M = \phi$" and "until $Q = \phi$" of the algorithm.
- lines 54, 55, & 56 perform step 8 of the algorithm. Steps 9, 10 and 11 of the algorithm are accomplished in lines 57 till 63. Note that the condition of step 9 is applied in line 57 because the first selected option of implementation is the one which contains only one hyperarc (hyperarc E_r, and thus "FirstOption->A->nxt == NULL") while all the rest contain at least 2 hyperarcs.
- "H" is a pointer to the first element in a linked list of options in which all G-paths attained so far are linked. Any generated option will be compared to all the options present in this list in order to verify if it has been generated

before, so that we can avoid examining it several times. If not, it will be added to this list.

- Lines 64 till 90 perform the task of generating G-paths from a selected one, as required in step 12 of the algorithm. Finally, the output is printed in lines 91, 92, and 93 by displaying the minimum cost option pointed by the pointer "K" along with its minimum cost in the variable "SMin", as well as the total number of G-paths generated.
- Note that in our code, we form two linked lists of type "ListOptions" representing options of implementation. The first is the one whose first element is pointed by "FirstOption" and which represents a list of options that are generated but not yet examined (because any completely examined option will be deleted from this list in line 90). As for the second, its first element is pointed by the pointer H and this list is formed to reserve all the options already generated, used only for comparison purposes.

List of Figures

Bibliography

[1] J. H. Reed, "*Software Radio: A modern Approach to Radio Engineering*", Prentice Hall PTR, 2002.

[2] PB. Kenington, "*Emerging technologies for software radio*", Electronics & Communication Engineering Journal April 1999; 69-73.

[3] *IEEE Communications Magazine, Special Issue on Software Radio,* vol. 37.1999.

[4] J. Mitola III, "*The software radio architectures*", IEEE Communications Magazine, vol. 33, pp. 26-38, May 1995.

[5] M. Woh, S. Seo, H. Lee, Y. Lin, S. Mahlke, T. Mudge, C. Chakrabarti, and F.K., "*The next generation challenge for Software Defined Radio*", Springer-Verlag Berlin Heidelberg, vol. LNCS 4599, pp. 343-354, 2007.

[6] J. Mitola III, "*Software Radio Architecture: Object Oriented Approaches to Wireless Systems Engineering."* John Wiley and Sons, Inc., 2000.

[7] IEEE Commun. Mag., vol. 37, no. 2, 1999, Special Issue on Software Radio.

[8] W.H.W.Tuttlebee, "*Software Defined Radio- Baseband Technology for 3G Handsets and Basestations [Book Review],*" Communications Engineer, vol.2, pp. 46-47, April-May 2004.

[9] A. Ivers and D. Smith, " *A practical approach to the implementation of multiple radio configurations utilizing reconfigurable hardware and software building blocks "*, in Proc. IEEE Military Communications Conference (MILCOM 97), vol. 3, pp. 1327-1332, IEEE, Monterey, Calif, USA, November 1997.

[10] A. Kountouris, C. Moy, L. Rambaud, and P. Le Corre, " *A reconfigurable radio case study: a software based multistandard transceiver for UMTS, GSM, EDGE and Bluetooth, in Proc"*. IEEE Vehicular Technology Conference (VTC 01), vol. 2, pp. 1196-1200, Atlantic City, NJ, USA, October 2001.

[11] O. Faust, B. Sputh, D. Nathan, S. Rezgui, A. Weisensee, and A. Allen, " *A single-chip supervised partial self-reconfigurable architecture for software defined radio* ", in Proc. 17th International Symposium on Parallel and Distributed Processing (IPDPS 03), IEEE, Nice, France, April 2003.

[12] H. Miranda, P. Pinto, and S. Silva, "*A self-reconfigurable receiver architecture for software radio systems"*, in Proc. IEEE Radio and Wireless Conference (RAWCON 03), pp. 241-244, IEEE, Boston, Mass, USA, August 2003.

[13] A. Pacifici, C. Vendetti, F. Frescura, and S. Cacopardi, "*A reconfigurable channel codec coprocessor for software radio multimedia applications*", in Proc. International Symposium on Circuits and Systems (ISCAS 03), vol. 2, IEEE, Bangkok, Thailand, May 2003.

[14] T. Hentschel and G. Fettweis, "*Sample rate conversion for software radio*", IEEE Commun. Mag., vol. 38, no. 8, pp. 142-150, 2000.

[15] W. Abu-Al-Saud and G. Stuber, "*Efficient sample rate conversion for software radio systems* " , in Proc. IEEE Global Telecommunications Conference (GLOBECOM 02), vol. 1, pp. 559-563, IEEE, Taipeh, Taiwan, Republic of China, November 2002.

[16] W. Abu-Al-Saud and G. Stuber, " *Modified CIC filter for sample rate conversion in software radio systems, IEEE Signal Processing Lett* "., vol. 10, no. 5, pp. 152-154, 2003.

[17] J.Ming, H. Y. Weng, and S. Bai, " *An efficient IF architecture for dual-mode GSM/W-CDMA receiver of a software radio, in Proc. IEEE International Workshop on Mobile Multimedia Communications*" (MoMuC 99), pp. 21-24, IEEE, San Diego, Calif, USA, November 1999.

[18] J. Dodley, R. Erving, and C. Rice, " *In-building software radio architecture, design and analysis, in Proc. IEEE 11th International Symposium on Personal, Indoor, andMobile Radio Communications (PIMRC 00)*", vol. 1, pp. 479-483, IEEE, London, UK, September 2000.

[19] W. Schacherbauer, A. Springer, T. Ostertag, C. Ruppel, and R. Weigel, " *A flexible multiband frontend for software radios using high IF and active interference cancellation* ", in Proc. IEEE MTT-S International Microwave Symposium Digest (IMS 01), vol. 2, pp. 1085-1088, IEEE, Phoenix, Ariz, USA, May 2001.

[20] A. Wiesler, " *Parameter gesteuertes Software Radio fur Mobilfunksysteme, Ph.D. dissertation, Forschungsberichte aus dem Institut fur Nachrichtentechnik, Universitat Karlsruhe (TH)*", Karlsruhe, Germany, May 2001.

[21] M. Beach, J. MacLeod, and P. Warr, "*Radio frequency translation for software defined radios*", in Software Defined Radio: Enabling Technologies, W. Tuttlebee, Ed., pp. 25-78, John Wiley & Sons, London, UK, 2002.

[22] G. Ahlquist, M. Rice, and B. Nelson, "*Error control coding in software radios: an FPGA approach, IEEE Personal Communications*" , vol. 6, no. 4, pp. 35-39, 1999.

[23] M. Valenti, " *An efficient software radio implementation of the UMTS turbo codec, in Proc. IEEE 12th International Symposium on Personal, Indoor, and Mobile Radio Communications*" (PIMRC 01), vol. 2, pp. G108-G113, IEEE, San Diego, Calif, USA, September 2001.

[24] V. Thara and M. Siddiqi, " *Power efficiency of software radio based turbo codec, in Proc. IEEE Region 10 Conference on Computers, Communications, Control, and Power Engineering (TENCON 02)*", vol. 2, pp. 1060-1063, IEEE, Beijing, China, October 2002.

[25] P. B. Kenington and L. Astier, " *Power consumption of A/D converters for software radio applications,*" IEEE Trans. Vehicular Tech.,vol.49, pp.643-650, March 2000.

[26] J. Singh, "*High speed analog-to-digital converter for software radio applications* ", in Proc. IEEE 11th International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC 00), vol. 1, pp. 39-42, IEEE, London, UK, September 2000.

[27] W. H. W. Tuttlebee, *"Software Defined Radio - Baseband Technology for 3G Handsets and Basestations,*" *Communications Engineer,* vol.2, pp. 46-47, April-May 2004.

[28] W. H. W. Tuttlebee, "*Software Defined Radio: Enabling Technologies"*, John Wiley and Sons Ltd. UK, 2002.

[29] W. H. W. Tuttlebee, " *Software-defined radio: facets of a developing technology*", IEEE Personal Communications, pp.38-44, April 1999.

[30] R. H. Walden, "*Performance trends for analog to digital converters*", IEEE Communications Magazine, vol. 37, pp. 96-101, February 1999.

[31] J. A. Wepman, " *Analog-to-digital converters and their applications in radio receivers*", IEEE communication magazines, pp. 39-45,May 1995.

[32] H. Tsurumi and Y. Suzuki, *Broadband RF stage architecture for software-defined radio in handheld terminal applications*", IEEE Communications Magazine, vol. 37, pp. 90-95, Feb. 1999.

[33] A. Hairapetian, "*An 81 MHz IF receiver in CMOS*, IEEE J. J. of Solid State Circuits, vol.31, pp. 1981-1996, December 1996.

[34] R. G. Vaughan, N. L. Scott, and D. R. White, "*The theory of bandpass sampling, IEEE Trans. Signal Processing*", vol. 39, no. 9, pp.1973 1984, Sept. 1991.

[35] F. Jondral, "*Parametrization- a technique for SDR implementation*", chapter 8 of Software-defined Radio Enabling technologies edited by W. Tuttlebee, Wiley 2002.

[36] Alaus L, Palicot J, Roland C, Louet Y and Noguet D, " *Promising Technique of Parameterization For Recongurable Radio, the Common Operators Technique : Fundamentals and Examples*" , march 2009, Revue Journal of Signal Processing Systems, Springer New York.

[37] Arnd-Ragnar Rhiemeier, *"Benefits and Limits of Parmetrized Channel Coding for Software Radio,"* in Proceedings of 2^{nd} Karlsruhe Workshop on Software Radios, Germany, March 2002.

[38] R. I. Lackey and D. W. Upmal. " *Speakeasy: the military software radio*". 33(5): 56-61, 1995. ISSN 0163-6804. doi: 10.1109/35.392998.

[39] R. Baines. The dsp bottleneck. 33(5): 46-54, May 1995. doi: 10.1109/35.392999.

[40] V. Rodriguez, C. Moy, J. Palicot, "*An optimal architecture for a multi-standard reconfigurable radio: Cost-minimizing common operators under latency constraints,*" *in* 15^{th} *European Information Society Technologies (IST) Summit, June 2006.*

[41] V. Rodriguez, C. Moy, J. Palicot, "*Install or Invoke?: The optimal trade-off between performance and cost in the design of multi-standard reconfigurable radios,*" *Wiley InterScience, Wireless Communications and Mobile Computing Journal, vol. 7, pp. 1143-1156, November 2007.*

[42] J. Palicot, C. Roland, *" FFT: a basic function for a Reconfigurable Receiver,* ICT, Feb. 2003, Papeete", Tahiti.

[43] A. Pacifici, C. Vendetti, F. Frescura, and S. Cacopardi, "*A reconfigurable channel codec coprocessor for software radio multimedia applications*". In *Proc. International Symposium on Circuits and Systems ISCAS*, volume 2, Chapter 2, pages 41-44, 25-28 May 2003. doi: 10.1109/ISCAS.2003.1205881.

[44] Alaus L, Noguet D, Palicot J," *A Common Operator Bank to Resolve Scheduling Issue on a Complexity Optimized SDR Terminal"*, Sixth Advanced International Conference on Telecommunications", AICT 2010 Barcelona, Spain,May 2010.

[45] Naoues, Malek; Noguet, Dominique; Louet, Yves; Grati, Khaled; Ghazel, Adel; ,"*An Efficient Flexible Common Operator for FFT and Viterbi Algorithms,*" Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd , vol., no., pp.1-5, 15-18 May 2011.

[46] A. Al Ghouwayel, Y. Louet, and J. Palicot, " *A reconfigurable architecture for the FFT operator in a software radio context,*" in *Proc. IEEE International Symposium on Circuits and systems ISCAS*, 2006

[47] L. Alaus, D. Noguet and J. Palicot, "*A Reconfigurable Linear Feedback Shift Register Operator for Software-defined Radios Terminal,*" ISWPC, Santorini, Greece, May 2008

[48] L. Alaus, D. Noguet, and J. Palicot, "*Extended Reconfigurable Linear Feedback Shift Register Operator for Software Defined Radio Terminal,*" ISSSTA Bologna, Italy, August 2008.

[49] C. Moy, J. Palicot, V. Rodriguez, D. Giri, "*Optimal Determination of Common Operators for Multi-Standard Software-Defined Radio,*" in Proceeding of 4^{th} Karlsruhe Workshop on Software Radios, Germany, March 2006.

[50] Sufi Tabassum Gul, Christophe Moy, Jacques Palicot: "*Graphical Modeling and Optimization of Air Interface Standards for Software-Defined Radios*" SCEE Research team, SUPELEC/ IETR, campus of Rennes 12^{th} IEEE International Multitopic conference-INMIC2008, karachi, Pakistan, December 2008.

[51] M. Naoues, L. Alaus, D. Noguet, "*A Common Operator for FFT and Viterbi algorithms*", The 13th Euromicro Conference on Digital System Design (DSD), 2010.

[52] Naoues M., Noguet D., Louët Y., Grati K., Ghazel A, "*An Efficient flexible common operator for FFT and Viterbi Algorithms* in Proceedings of the C2POWER Workshop - VTC 2011 - C2POWER Workshop - VTC 2011, Hongrie (2011) [hal-00657436 - version 1]

[53] S. T. Gul, C. Moy and J. Palicot, "*Two Scenarios of Flexible Multi-Standard Architecture Designs using a Multi-Granularity Exploration",* in Proceeding IEEE 18^{th} International Symposium on Personal, Indoor and Mobile Radio Communications PIMRC, pp. 1-5, 3-7 Sept. 2007.

[54] R. Timothy Marler. "*A Study of Multi Objective Optimization Methods for Engineering Applications*", PhD thesis, The University of Iowa.

[55] R. Kahraman and M. Sunar , "*A comparative study of multiobjective optimization methods in structural design*" ., Turkish Journal of Engineering and Environmental Sciences 25: 69-78, 2001.

[56] I. Y. Kim and O. De Weck , " *Adaptive weighted sum method for bi-objective optimization*" Structural and Multidisciplinary Optimization 29.2:149-158, 2005.

[57] P. Korhonen .1998), "*Multiple objective programming support. Technical Report*" IR-98-010, International Institute for Applied Systems Analysis, Laxenburg, Austria.

[58] L. Zadeh, " *Optimality and non-scalar-valued performance criteria*", IEEE Trans. Autom. Control 8 (1963) 5960.

[59] Suppapitnarm et al, *"Heuristicly refined multiobjective random search"*, Songklanakarin J. Sci. Technol., 2005, 27(2): 301-312.

[60] Sufi T. Gul, "*Optimization of Multi-Standards Software Defined Radio Equipments : A Common Operator's Approach"*, PhD thesis, november 2009.

[61] L.R. Foulds, " *Graph Theory Applications",* 1992 Springer-Verlag New York, Inc.

[62] J.A Bondy & U.S.R. Murty, "*Graph Theory, Graduate Texts in Mathematics*", Springer 2008.

[63] Garey, M.R. and D.S Johnson: "*Computers and Intractability: A Guide to the Theory of NP-completeness"*, W.H.Freeman, San Francisco, CA (1979).

[64] J.C. Bermond and F.O. Ergincan, "*Bus Interconnection Networks"*, Discrete Applied Mathematics, 68:1-15, 1996.

[65] J.C. Bermond, R. Dawes, and F.O. Ergincan, "*De Bruijn and Kautz bus networks"*, Networks, 30:205-218, 1997.

[66] V. Voloshin, " *Coloring Mixed Hypergraphs: theory, algorithms and applications*". AMS, Providence, 2002.

[67] V. Voloshin, "*Introduction to Graph and Hypergraph theory*", New York, 2009.

[68] C. Berge: "*Hypergraphs: combinatorics of finite sets*", North-Holland Amsterdam, 1989.

[69] G. Gallo, G. Longo, S. Pallottino, and Sang Nguyen, "D*irected hypergraphs and applications. Discrete applied mathematics"*, 1993.

[70] Lars Relund Nielsen, Daniele Pretolani: " *A remark on the definition of B-hyperpath*", 2001.

[71] Narsingh DEO "*Graph Theory with Applications to Engineering and Computer Science"*, 1974 by Prentice-Hall, Inc., Englewood Cliffs, N. J.

[72] G. Brassard and P. Bratley, *" Algorithmics: Theory and Practice*, Prentice Hall Englewood Cliffs", 1^{st} Edition, New Jersey, 1988.

[73] LIN, S., "*Computer Solution of the Travaling Salesman Problem",* BSTJ, Vol. 44, 1965, 2245-2269.

[74] Kruskal, J. B., Jr,. "*On the Shortest Spanning Subtree of Graph and the Traveling Salesman Problem*", Proc. Am. Math. Soc., Vol. 7, 1956, 48-50.

[75] Prim, R. C., "*Shortest Connection Networks and Some Generalizations"*, *Bell System Tech. J.,* Vol. 36, Nov. 1957, 1389-1401.

[76] Held, M. and R. M. Karp, "*The Traveling-Salesman Problem and Minimum Spanning Trees: Part II*", *Mathematical programming,* Vol. 1, 1971, North-Holland Publishing Company, 6-25.

[77] Tarjan, R., "*Depth-First Search and Linear Graph Algorithms," SIAM J. Comput.,* Vol. 1, No. 2, June 1972, 146-160.

[78] Dijkstra, E. W., "*A note on two problems in connection with Graphs,*" *Numerische Math,* Vol. 1, 1959, 269-271.

[79] Dreyfus, S. E., "*An Appraisal of some Shortest-Path Algorithms,*" *J. Operations Research,* Vol. 17, No. 3, 1969, 395-412.

[80] Dantzig. G. B, "*All Shortest routes in a graph,*" *Proceedings of the International Symposium on Graph theory,* Rome, Italy, July 1966, 91-92 Published by Dunod Editeur, Paris.

[81] Floyd. R. W., "*Algorithm 97: Shortest path,*" *Comm, ACM,* Vol. 5, 1962, 345.

[82] J. L. Gross and J. Yellen, "*Handbook of Graph Theory (Discrete Mathematics and its Applications)*". CRC Press, New York, USA., 2004.

[83] F. Havet, "channel assignment and (weighted coloring)," Ubiquitous Networks - algorithms for telecommunication.

[84] C. McDiarmid. "*On the span in channel assignment problems: bounds, computing and counting*", *Discrete Math.* 266:387-397, 2003.

[85] Italiano , G.F. and U. Nanni, "*On line maintenance of minimal directed hypergraphs*, Proc. 3◦ Convegno Italianodi Informatica Teorica, Mantova, World Science Press (1989), 335-349.

[86] Jeroslow, R.G., R.K. Martin, R.R. Rardin and J. Wang, "*Gainfree Leontiev flows problems*, Tech. Rept., School of Business, University of Chicago (1989).

[87] Cook, S., "*The complexity of theorem-proving procedures*", Proc. 3-th ACM Symp. on Theory of Computing (1971), 151-158.

[88] P. Maatouk Kaiser, Y. Louët, A. El Sahili, *"A cost function expression for SDR multi-standard systems design using directed hypergraphs"*, *URSI, Istanbul, Turkey*, August 2011.

[89] P. Maatouk Kaiser, A. El Sahili, Y. Louët, *"An upper bound for the total number of options to implement an SDR multi-standard system", International Conference on Telecommunications (ICT), Beirut, Lebanon*, April 2012.

[90] P. Maatouk Kaiser, A. El Sahili, Y. Louët, "Complexity analysis for an optimization problem of an SDR multi-standard system", *in preparation.*

[91] Dowling, W. and J. Gallier, "*Linear-time algorithms for testing the satisfiability of propositional Horn formulae,*" J. of Logic Programming 3 (1984), 267-284.

[92] Itai, A. and J. Makowsky, "*On the complexity of Herbrand's theorem*", Tech. Rept. 243, Dept. Comp. Sci., Israel Inst. of Technology (1982).

[93] R. Diestel, "*Graph theory*", Springer-Verlag Heidelberg, New York 2006.

[94] Ausiello, G., A. D'Atri and D. Saccà, "*Strongly equivalent directed hypergraphs,* in: Analysis and Design of Algorithms for combinatorial problems," Annals of Discrete Mathematics, 25 (1985), 1-25.

[95] Ausiello, G., A. D'Atri and D. Saccà, "*Minimal representation of directed hypergraphs,*" SIAM J. Comput., 15 (1986), 418-431.

[96] Ausiello, G., G.F. Italiano and U. Nanni, "*Dynamic maintenance of directed hypergraphs,*" Theor. comp. Sci. 72 (1990), 97-117.

[97] Nguyen, S. and S. Pallottino, "*Equilibrium traffic assignment for large scale transit network,*" Eur. J. of Oper. Res., 37 (1988), 176-186.

[98] Nguyen, S. and S. Pallottino, "*Hyperpaths and shortest hyperpaths,* in: Combinatorial Optimization (B. Simeone, ed.)," Lecture notes in mathematics, 1403, Springer-Verlag, Berlin (1989), 258-271.

[99] Hopcroft, J. E., and J. D. Ullman, "*Formal languages and their relation to Automata,*" Addison-Wesley, Reading, MA (1.3; 2.2; 2.3; 7.4; 7.5; 7.6; A4.2; A10.1; A10.2), 1969.

[100] Aho, A. V., J. E. Hopcroft, and J. D, Ullman, "*The design and analysis of computer algorithms* " Addison-Wesley, Reading, MA (1.3; 2.3; 4.0; 6.1; 7.4), 1974.

[101] Cook, S. A., "*The complexity of theorem-proving procedures*", Proc. 3^{rd} Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery, New York, 151-158 (1.5; 2.6; 3.1.1; 5.2; A1.4; A9.1), 1971.

[102] P. Maatouk Kaiser, A. El Sahili, Y. Louët, *"An optimization algorithm for SDR multi-standard systems using Directed Hypergraphs", Frequenz, issn 2191-6349 vol. 66, issue 9-10, pp. 251-260, 2012 ; [hal 00735069].*

[103] B. Hajek, "*A tutorial survey of theory and applications of simulated annealing,"* 24th IEEE Conference on Decision and Control, Dec 1985, pp.755-760.

[104] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *" Equations of state calculations by fast computing machines,"* Journal of Chemical Physics, vol. 21, pp. 1087-1092, 1953.

[105] M. Pincus, *"A Monte Carlo method for the approximate solution of certain types of constrained optimization problems,"* Opertaions Research, vol. 18, pp. 1225-1228, 1970.

[106] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *"Optimization by Simulated Annealing,"* Science Journal, vol. 220, pp. 671-680, May 1983.

[107] S. Russell and P. Norvig, *"Artificial Intelligence: A Modern Approach",* Prentice Hall, Upper Saddle River, New Jersey, 1995.

Publications

International Journal Papers

1. P. Maatouk Kaiser, A. El Sahili, Y. Louët, "An Optimization Algorithm for SDR Multi-Standard systems Using Directed Hypergraphs", *Frequenz, issn 2191-6349 vol. 66, issue 9-10, pp. 251-260, 2012 ; [hal 00735069].*
2. P. Maatouk Kaiser, A. El Sahili, Y. Louët, "Complexity analysis for an optimization problem of an SDR multi-standard system", *in preparation.*
3. P. Maatouk Kaiser, A. El Sahili, Y. Louët, "A program code for a Minimum Cost Design algorithm of SDR multi-standard systems using graph theory", *in preparation.*

International Conference Papers

1. P. Maatouk Kaiser, Y. Louët, A. El Sahili, "A cost function expression for SDR multi-standard systems design using directed hypergraphs", *URSI, Istanbul, Turkey*, August 2011.
2. P. Maatouk Kaiser, A. El Sahili, Y. Louët, "An upper bound for the total number of options to implement an SDR multi-standard system", *International Conference on Telecommunications (ICT), Beirut, Lebanon*, April 2012.

International Conference Workshops

1. P. Maatouk Kaiser, S. T. Gul, C. Moy, Y. Louët, "Graph theory approach for optimization of multi-standards software defined radio equipements", *European Reconfigurable Radio Technologies (ERRT) Conference, Mainz*, June 2010.
2. P. Maatouk Kaiser, Y. Louët, A. El Sahili, "An algorithm proposal for a minimum cost SDR multi-standard system using graph theory", *Workshop on Software Radio, Karlsruhe, Germany*, March 2012.
3. P. Maatouk Kaiser, Y. Louët, A. El Sahili, "An NP optimization problem related to a software-defined radio (SDR) design", *DIMACOS, Lebanese University â Hadath Lebanon*, November 2012.

The Software-Defined Radio (SDR) concept is emerging as a potential and efficient solution for designing flexible future-proof multi-standard systems. A way of realizing a multi-standard terminal is to identify the appropriate common functions and operators inside and between the standards. This is what's called the parametrization approach, which can be divided into two categories: the pragmatic approach which is a practical version to create and develop common operators, and the theoretical approach which represents a graphical exploration of the SDR multi-standard system at different levels of granularity accompanied with an optimization problem. It's in this last approach where our thesis subject dwells. In this context, a suggested cost function (in previous work) has to be optimized in order to select the convenient common operators between the different standards, enabling to construct an optimal design. In our work, we theoretically model a previously proposed graph structure of an SDR multi-standard system as a directed hypergraph as well as provide an alternative mathematical formal expression of the suggested cost function, using various graph theoretical definitions and notations. Afterwards, we prove that the associated optimization problem is an NP-problem under a certain constraint, which entails a proof of exclusion of some particular design options when searching for a minimum cost design. This was the second contribution in this thesis before we finally present a new algorithm (which exploits various modelization aspects of directed hypergraphs) that can solve the optimization problem, whose interest is in it giving an exact-optimal solution to our problem instead of a near-optimal one provided by heuristics. A program code for this algorithm was developed in C-language, and then it was applied on several generic case examples in order to explore its performance skills.

Keywords: Software-defined Radio, Graph Theory, Directed Hypergraph, Optimization, NP-problem, Minimum Cost Design Algorithm, Computational Complexity.

Le concept de radio logicielle (SDR) apparaît comme une solution pertinente pour concevoir des équipements multi-standards. Une façon de réaliser de tels équipements est d'identifier les fonctions et opérateurs communs entre les standards. Cette approche s'appelle la paramétrisation, qui peut être divisée en deux catégories : l'approche pragmatique qui est une version pratique pour créer et développer des opérateurs communs à partir d'opérateurs existants, et l'approche théorique dont l'objectif est de réaliser une exploration graphique d'un équipement multi-standards selon différents niveaux de granularité, accompagnée d'un problème d'optimisation. C'est cette dernière approche qui a constitué le sujet de base de cette thèse. Dans ce contexte, une fonction de coût doit être optimisée afin de sélectionner les opérateurs communs entre les différentes normes, ce qui permet de proposer une configuration optimale à partir de laquelle sont déduits les opérateurs communs. Dans notre travail, nous avons dans un premier temps modélisé théoriquement la structure graphique d'un système multi-standards par un hypergraphe orienté. En outre, nous avons fourni une expression mathématique alternative de la fonction de coût suggérée, en utilisant des définitions propres à la théorie des graphes. Ensuite, nous avons montré que le problème d'optimisation associé était un problème NP sous une certaine contrainte, ce qui a entraîné une preuve d'exclusion de certaines configurations dont les coûts ne peuvent être minimaux. Ceci a constitué la deuxième contribution de cette thèse. Enfin, nous avons proposé un nouvel algorithme (exploitant les concepts des hypergraphes orientés) permettant de résoudre le problème d'optimisation donné, et dont l'intérêt est de donner une solution optimale du problème au lieu d'une solution approchée fournie par les méthodes heuristiques classiques. Un programme associé à cet algorithme a été développé en langage C, puis appliqué à plusieurs exemples de cas génériques afin d'en étudier les performances.

Mots-clés : La Radio Logicielle Restreinte (SDR), La théorie des graphes, hypergraphe orienté, optimisation, problème NP, algorithme de configuration à coût minimal, complexité de calcul.