UNIVERSITE PARIS VI - PIERRE ET MARIE CURIE

U.F.R. D'INFORMATIQUE

THESE DE DOCTORAT

En vue de l'obtention du grade de
docteur de l'université Paris VI
en informatique

Présentée par

Timothée BOSSART

Modèles pour l'optimisation de la simulation au cycle près de systèmes synchrones

Soutenue le 27 novembre 2006 devant le jury composé de :

M.	Jens GUSTEDT	rapporteur
M.	Aziz MOUKRIM	rapporteur
M^{me}	Nathalie DRACH-TEMAM	examinatrice
M.	Olivier HUDRY	examinateur
M^{me}	Alix MUNIER-KORDON	directrice de thèse
M.	Eric SANLAVILLE	examinateur
M.	Francis SOURD	examinateur

UNIVERSITE PARIS VI - PIERRE ET MARIE CURIE

U.F.R. D'INFORMATIQUE

PHD THESIS

Timothée BOSSART

Models for the Optimization of Cycle-based Synchronous System Simulation

Defense date : November 27^{th}, 2006

Committee in charge :

Mr	Jens GUSTEDT	reporter
Mr	Aziz MOUKRIM	reporter
Mrs	Nathalie DRACH-TEMAM	examiner
Mr	Olivier HUDRY	examiner
Mrs	Alix MUNIER-KORDON	advisor
Mr	Eric SANLAVILLE	examiner
Mr	Francis SOURD	examiner

Aux Jouan :-)

"Why, sometimes I've believed as many as six impossible things before breakfast."
Lewis Carroll

Remerciements

Je tiens à exprimer ici ma reconnaissance à toutes les personnes qui m'ont aidé, encouragé ou tout simplement soutenu tout au long de mon travail de thèse. En premier lieu, le département SOC (ASIM) du laboratoire d'informatique de Paris 6 pour avoir soutenu mes travaux, et l'UFR de sciences de l'université Paris 12 Créteil Val-de-Marne pour avoir pris le relais lors des derniers mois en m'offrant un poste d'ATER.

Je voudrais également remercier Jens Gustedt et Aziz Moukrim d'avoir accepté d'évaluer mon manuscrit. J'ai particulièrement apprécié vos rapports détaillés et approfondis témoignant de l'intérêt que vous lui avez porté. Merci aussi à Nathalie Drach-Temam, Olivier Hudry, Eric Sanlaville et Francis Sourd pour avoir accepté de faire partie du jury de la soutenance. Merci à Alix Munier Kordon pour m'avoir confié ce sujet de thèse et pour avoir dirigé mes travaux pendant 4 ans.

A titre personnel, je remercie Jean-Marc Vincent pour m'avoir (gentiment) forcé à m'inscrire en DEA il y a quelques années. Ma vocation pour la recherche est née de cette époque, où j'ai rencontré Denis Trystram et Gilles Parmentier lors de mon stage. Merci aussi à eux.

Merci à tous les gens qui m'ont soutenu directement ou indirectement durant ces années. Plutôt que d'en oublier, je préfère citer les Jouan, les Trignat, les Braunstein, et mes potes du go, les autres se reconnaitront. De toute façon ils savent à quel point je leur suis reconnaissant !

... et puis merci à Cécile de m'avoir supporté, c'est bientôt ton tour, alors bonne chance :-)

Résumé

Cette thèse traite de fonctions de coûts de graphes orientés sans circuit destinées à modéliser simplement la mémoire d'une machine lors de l'exécution d'un programme. L'objectif final est d'accélérer le fonctionnement de programmes de simulation au cycle près de systèmes synchrones en améliorant la gestion de la mémoire lors de leur exécution.

Dans cette thèse, nous définissons deux fonctions de coût, appelées DSC (Directed Sum Cut) et UCS (Uniform Cost Stack), ainsi que les problèmes d'optimisation correspondants. Nous montrons tout d'abord que l'obtention d'une solution optimale est un problème difficile pour les deux fonctions, dès les graphes de profondeur deux. Nous présentons ensuite des algorithmes permettant d'obtenir des solutions optimales en temps polynomial pour certaines classes de graphes : les anti-arborescences, les arborescences, certains graphes série-parallèles, et finalement les ordres intervalles. Nous concluons en présentant les résultats d'expériences pratiques montrant que nos modèles permettent d'obtenir des accélérations de temps de simulation.

Abstract

This thesis deals with directed acyclic digraphs cost criteria designed to model the memory of a machine during the execution of a program. The aim is to speedup synchronous system cycle-based simulation programs by increasing memory management during their execution.

In this thesis, we define two criteria, called DSC (Directed Sum Cut) and UCS (Uniform Cost Stack), along with the corresponding optimization problems. We first show that both problems are difficult for digraphs of depth two. We then present polynomial algorithms leading to optimal solutions for some classes of digraphs : intrees, outtrees, some series-parallel graphs, and interval orders. We conclude by showing experimental results in which heuristics designed for our models lead to significant simulation speedups.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

Pour des raisons de lisibilité, la signification d'une abréviation ou d'un acronyme n'est souvent rappelée qu'à sa première apparition dans le texte d'un chapitre. Par ailleurs, puisque nous utilisons toujours l'abréviation la plus usuelle, il est fréquent que ce soit le terme anglais qui soit employé, auquel cas nous présentons une traduction.

Chapitre 1

CAO	Computer Aided Design (CAD)	Conception Assistée par Ordinateur
SoC	System on Chip	Système intégré
NoC	Network on Chip	Réseau intégré
ALU	Arithmetic Logic Unit	Unité arithmétique et logique

Chapitre 2

LRU	Least Recently Used	utilisée le moins récemment
LFU	Least Frequently Used	utilisée le moins fréquemment
DSC	Directed Sum Cut	Somme Coupe Orientée (inusité)
UCS	Uniform Cost Stack	Pile à Coût Uniforme

Chapitre 3

SP	Series-Parallel	graphes série-parallèles
2TSP	2-Terminal Series-Parallel	graphes série-parallèles 2-terminaux
r-2TSP	Restricted 2-Terminal Series-Parallel	graphes série-parallèles 2-terminaux restreints

Liste des notations

Nous avons regroupé ci-dessous les principales notations employées dans les différents chapitres du document. Dans la mesure du possible, nous avons tenté de conserver les mêmes notations d'un chapitre à l'autre. Nous présentons tout d'abord une liste générale puis des listes relatives aux différents chapitres. On notera que seules les notations qui diffèrent de celles précédemment définies seront données dans ces listes. Enfin, certaines notations, apparaissant uniquement de manière ponctuelle, ont été omises.

Notations générales

$G = (V, A)$	graphe orienté sans circuit, dont l'ensemble des sommets est V et dont l'ensemble des arcs est A
$n = \|V\|, m = \|A\|$	nombre de sommets et d'arcs de G
$\Gamma^+(u)$	ensemble des successeurs de $u \in V$
$\delta^+(u) = \|\Gamma^+(u)\|$	degré sortant de u
$\Gamma^-(u)$	ensemble des prédécesseurs de $u \in V$
$\delta^-(u) = \|\Gamma^-(u)\|$	degré entrant de u
$\varphi : V \mapsto \{1, \cdots, n\}$	une numérotation des sommets de V
$\mathrm{DSC}(\varphi, G)$	coût DSC de la numérotation φ pour G
$\mathrm{UCS}(\varphi, G)$	coût UCS de la numérotation φ pour G

Chapitre 2

$\mathcal{C}(\varphi, (u, v))$	coût DSC de la numérotation φ pour l'arc (u, v)
$\mathcal{C}(\varphi, u)$	coût DSC de la numérotation φ pour le sommet u
$\mathcal{C}(\varphi, G)$	coût DSC de la numérotation φ pour G
LD,RD,OP	opérateurs de manipulation de la pile pour le modèle UCS
θ_u	ordre d'empilement pour le sommet u
$\mathrm{UCS}_\theta(\varphi, G)$	coût de θ pour φ sur G

Introduction

Lors de la conception de circuits ou de systèmes intégrés, l'aide de l'ordinateur est devenue prépondérante. Le cercle vertueux engendré par le fait que les microprocesseurs d'une génération aident à la conception de ceux, plus puissants, de la génération suivante, est connu sous le nom de *loi de Moore*. Cette loi, énoncée en 1965 par Gordon Moore [Moo65], cofondateur de la société Intel, affirme que "le nombre de transistors par circuit de même taille va doubler tous les 18 mois" (voir figure 1). Même si actuellement le modèle converge vers une période de 24 mois (comme l'a corrigé Moore lui-même en 1975 [Moo75]), la loi a marqué les esprits puisqu'elle est devenue un défi pour les concepteurs de microprocesseurs.

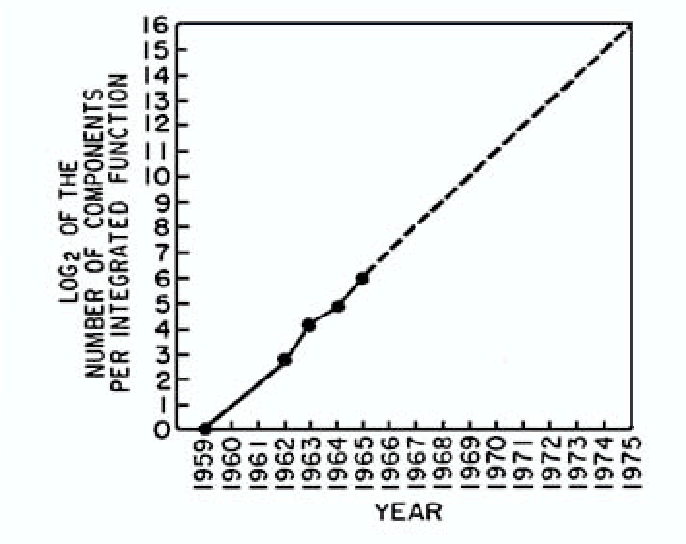


FIG. 1 – Courbe tirée de l'article de Gordon Moore (1965)

Face à une telle croissance, la complexité des problèmatiques de la conception assistée par ordinateur augmente de la même façon. Le cadre dans lequel se placent les travaux de cette thèse est celui de la *simulation*. En effet, partant du constat qu'un programme est beaucoup plus facile à modifier qu'une puce de silicium, les concepteurs ont rapidement réalisé qu'il était nécessaire de disposer d'un moyen permettant, à partir d'une description d'un système, de prévoir son comportement final *pendant sa conception*. Ceci a été rendu possible par l'élaboration de programmes de simulation, permettant d'éliminer un maximum de problèmes avant de procéder à des tests réels. Encore une fois, la croissance de la complexité des entrées a rendu obsolètes les méthodes les plus simples, et la simulation

est devenue un important goulot d'étranglement pour la conception de systèmes intégrés. Aujourd'hui, les deux méthodes extrêmes présentées dans le chapitre 1 cohabitent en fonction des différents objectifs et compromis que le concepteur veut fixer. Il s'agit d'une part de la *simulation événementielle*, qui réalise un ordonnancement dynamique du fonctionnement des composants du système, et d'autre part de la *simulation au cycle près*, qui repose sur un ordonnancement statique par la création d'un programme de simulation.

Tout d'abord, la *simulation événementielle* consiste en un ordonnancement dynamique des calculs à effectuer, en fonction des modifications de ceux dont ils dépendent. L'avantage de cette méthode est d'une part de ne pas faire répétitivement des calculs dont les paramètres n'ont pas été modifiés depuis le dernier passage, et d'autre part de donner la possibilité d'effectuer des simulations avec une discrétisation temporelle paramétrable. Cette liberté est obtenue au prix de la gestion de listes et d'échéanciers. A l'inverse, la *simulation au cycle près* procède par un ordonnancement statique, et construit un programme de simulation, qui est ensuite compilé puis exécuté. Libérée des aspects dynamiques, cette méthode est plus rapide que la simulation événementielle, mais elle est en général limitée à la simulation de systèmes *synchrones*, c'est-à-dire pour lesquels tous les composants sont synchronisés sur la même horloge.

Les systèmes actuels résultant de l'assemblage de composants de moindre complexité, on en est venu à dire qu'ils sont localement synchrones, et globalement asynchrones, ce qui permet maintenant d'appliquer des méthodes hybrides pour leur simulation [SS92]. Or, les parties synchrones sont encore suffisamment complexes pour résister aux méthodes classiques de simulation au cycle près, puisque les programmes générés ne sont pas adaptés aux optimisations classiques des compilateurs. En particulier, le grand nombre de variables qu'ils utilisent engendrent des problèmes de gestion de la mémoire lors de l'exécution, dans lesquels réside le cœur des travaux de cette thèse. La dernière section du chapitre 1 est consacrée à un état de l'art succinct sur les méthodes de simulation au cycle près.

Les deux modèles proposés dans le cadre de nos travaux (DSC, pour Directed Sum Cut et UCS, pour Uniform Cost Stack, et présentés au chapitre 2), ont pour objectif, à partir de la description d'un système, de donner une estimation de la durée d'exécution du programme final sur un ordinateur. Ces coûts reposent sur une estimation des durées de chargement des variables du programme à partir de la mémoire. La simplicité des modèles nous permet ensuite de formaliser deux problèmes d'optimisation combinatoire de la classe des problèmes de *numérotation de graphes*.

La section 2.1 du chapitre 2 est dédiée à la justification des fondements de nos modèles. Les deux modèles partent du fait que les différentes politiques de gestion de mémoire des ordinateurs tentent de prévoir les futures utilisations des variables en repoussant celles qui sont peu utilisées dans des zones difficiles d'accès, de façon à garder celles qui le sont souvent dans des zones peu coûteuses. Le modèle DSC, présenté en section 2.2.3, repose sur le constat qu'une numérotation d'un graphe correspond à une numérotation des lignes du programme correspondant, et vise donc à modéliser le fait que les utilisations d'une variable doivent être faites dans la mesure du possible le plus rapidement après sa création.

Le modèle UCS, présenté en section 2.2.4 part d'une représentation de la mémoire sous forme d'une pile, dans laquelle les accès aux variables ont un coût plus important selon leur date de dernière utilisation.

Dans le chapitre 3, dédié à la présentation des résultats théoriques obtenus lors de nos travaux, nous montrons que les deux problèmes sont difficiles pour les graphes de profondeur 2. Ensuite nous montrons que l'étude de certaines classes de graphes peut conduire à des sous-problèmes résolvables en temps polynomial. Tout d'abord, en sections 3.2.1) et 3.2.2, nous montrons que la structure récursive des anti-arborescences et des arborescences permet d'obtenir une résolution polynomiale. De la même façon, nous présentons en section 3.2.3 un algorithme exploitant la structure récursive d'une classe de graphes série-parallèles pour atteindre une numérotation optimale en temps polynomial. Nous montrons aussi dans cette section que la méthode n'est pas valable pour d'autres classes de graphes série-parallèles. Le dernier résultat de polynomialité, présenté en section 3.2.4, concerne les ordres intervalles. Ce résultat repose sur une propriété des sommets sans successeurs de tels graphes.

Le chapitre 4 est consacré à la présentation de résultats expérimentaux. Dans un premier temps, nous présentons différentes heuristiques élaborées pour nos études, puis nous montrons que malgré leur simplicité, les deux modèles de mémoire conduisent à des estimations cohérentes des durées d'exécution des programmes. Ensuite, nous montrons que nos travaux ont mené à des heuristiques permettant d'obtenir des programmes plus rapides que les méthodes actuellement utilisées.

Le dernier chapitre présente finalement les conclusions de nos travaux, ainsi que les perspectives de recherche qui en découlent, avec en particulier l'intégration complète de nos méthodes dans des environnements de simulation, ainsi que l'étude de classes de graphes pour lesquelles des algorithmes polynomiaux n'ont pas encore été trouvés.

Chapitre 1

Simulation

Un des problèmes lors de la conception de circuits ou de systèmes intégrés est que les modifications a posteriori sont difficiles, voire impossibles dans le cas du silicium. On estime d'ailleurs que le coût de la détection d'une faute (le coût de la réparation est prépondérant) dans un circuit est multiplié par un facteur 10 à chacune des étapes suivantes : conception, test, fabrication, fonctionnement dans la machine finie. Au contraire, tout programme fonctionnant sur un ordinateur est relativement facile à modifier. Le processus par lequel on émule le fonctionnement d'un système par le fonctionnement d'un programme sur un ordinateur est appelé simulation.

Avec la croissance exponentielle de la taille et de la complexité des circuits, la simulation est devenue un important goulot d'étranglement dans la CAO (*Conception Assistée par Ordinateur*), et il est important de comprendre qu'un compromis doit être accepté par le concepteur entre ses différents objectifs (détection de fautes, génération de jeux de test, exactitude logique, respect de contraintes temporelles, etc...). En conséquence, lors de la conception de grands et très grands systèmes intégrés, l'utilisation de la simulation a pris une telle place, que toute méthode permettant une accélération est devenue primordiale. Après avoir détaillé le contexte de la conception de systèmes intégrés et les différents buts de la simulation, nous montrerons dans ce chapitre deux méthodes ayant chacune des objectifs disctincts : la simulation événementielle, et la simulation au cycle près. Nous conclurons pour chacune en montrant les limites des méthodes actuellement utilisées.

1.1 Contexte de la conception de systèmes intégrés

L'objectif de cette section est de présenter le cadre dans lequel se situe la simulation : la conception de systèmes intégrés. Les méthodes et les technologies permettant l'intégration de circuits numériques sur des puces de silicium n'ont cessé d'évoluer. En particulier, la

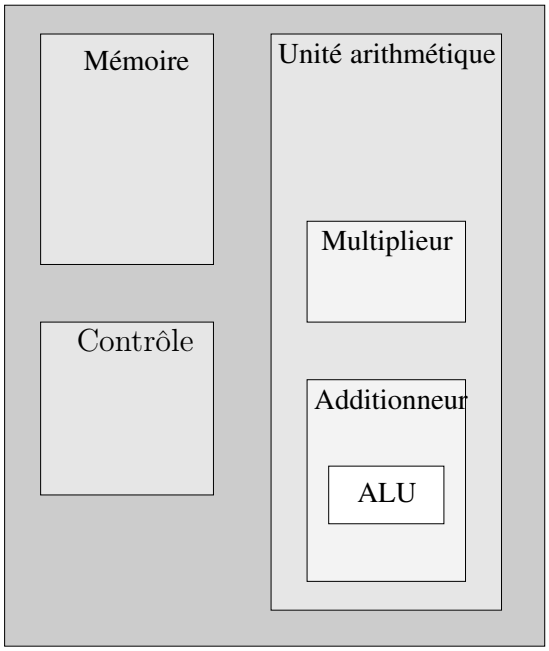


Fig. 1.1 – Description hiérarchique d'un système

densité actuelle permet maintenant d'intégrer sur une seule puce des systèmes entiers (SoC, pour *System on Chip*), c'est à dire un microprocesseur avec des mémoires, des bus, et tous les composants nécessaires à une application. On peut même concevoir des réseaux de processeurs sur puce (NoC, pour *Network on Chip*). Bien évidemment, l'aide de l'ordinateur est primordiale pour la plupart des étapes de la conception de tels systèmes. C'est la CAO. La simulation (voir section 1.2) est une de ces étapes.

1.1.1 Description d'un système et méthodes de conception

En général, un concepteur de systèmes intégrés obtient d'un client des *spécifications* du système requis. Il commence par le diviser en *blocs fonctionnels*, c'est à dire une description de haut niveau (figure 1.1). Chacun de ces blocs peut alors être spécifié plus finement. En bas de l'échelle d'abstraction, le concepteur arrive finalement à écrire des circuits, c'est à dire des ensembles de portes logiques reliées entre elles par des fils. On peut aussi appeler ces fils des *signaux*, en les représentant par leur valeur. Cette méthode de conception est appelée *top-down*. A l'inverse, lorsqu'il prend connaissance relativement tôt qu'il devra utiliser certaines briques de base (comme par exemple les ALU, pour Unités de Logique Arithmétique), le concepteur peut commencer son travail par leur élaboration, et ensuite les assembler progressivement entre elles de façon à monter de niveau d'abstraction. Il suit alors une méthode de type *bottom-up*. Le choix des algorithmes de simulation dépendent évidemment beaucoup du niveau d'abstraction auquel se place le concepteur.

1.2 Objectifs et méthodes de la simulation

L'objectif de cette section est de présenter la simulation dans le cadre de la CAO au travers de ses différents objectifs, ainsi que des différents niveaux appropriés. On présentera en détails les deux méthodes extrêmes en terme de complexité : la simulation événementielle et la simulation au cycle près. Cette section synthétise des introductions présentées par Gosling [Gos93], ainsi que Hommais [Hom01].

1.2.1 Objectifs de la simulation

Les objectifs de la simulation sont multiples, et on montre dans la suite que le concepteur de systèmes doit élaborer sa stratégie de simulation en les prenant en compte. Nous présentons ici les quatre principaux.

Exactitude fonctionelle

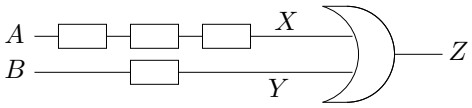
L'élément qui intéresse le plus le concepteur lors de la simulation est a priori de vérifier que le système effectue correctement la fonction pour laquelle il a été conçu. Pour cela, deux points doivent être considérés. Tout d'abord, les spécifications du système doivent être connues sans ambiguïté, de façon à ce que la fonction correcte soit comparable avec la sortie de la simulation. De plus, le comportement du système sous des conditions inhabituelles peut être important. Si ces conditions ne font pas partie de la spécification, la simulation de ce comportement peut se révéler très difficile.

Respect des contraintes temporelles

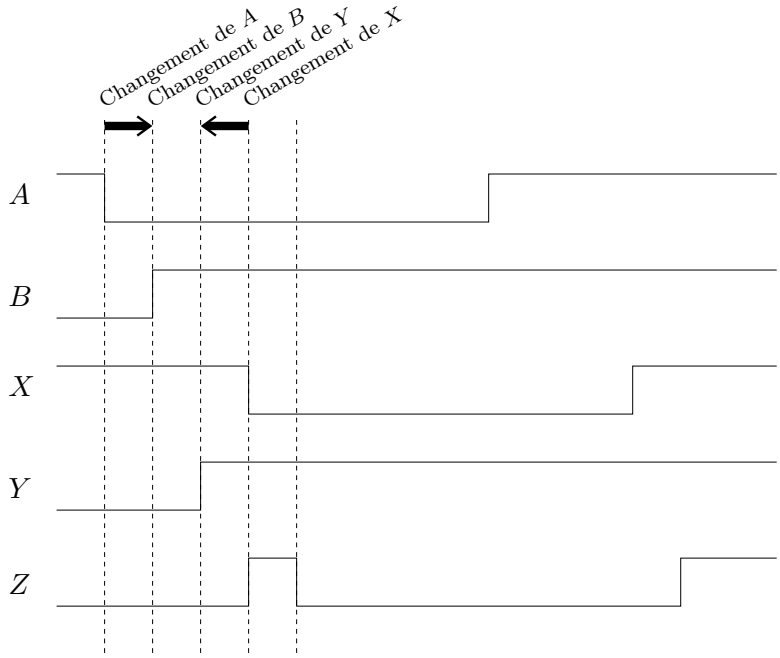
Lorsque le système final est succeptible d'être soumis à des contraintes temporelles, et même lorsque ces contraintes paraissent très lâches, il peut être nécessaire de vérifier qu'il les respecte. Par exemple, si chaque porte logique induit un certain délai, un système peut rapidement subir un délai non négligeable [Gos93].

Détection de fautes

Les délais divers dans un système peuvent induire des accidents (c'est à dire un signal dont la valeur ne respecte pas la logique, voir l'exemple de la figure 1.2, explicité ci-dessous) qui modifient radicalement les calculs. Le concepteur peut vouloir vérifier ce genre d'événements, et attendre différents comportements de la part du simulateur, car



(a) Circuit

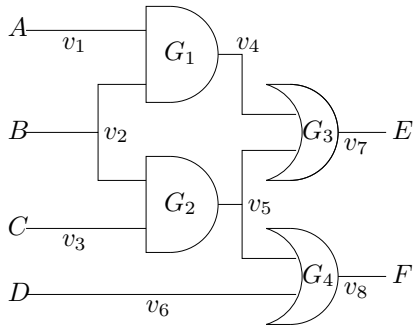


(b) Evolution des signaux au cours du temps

Fig. 1.2 – Exemple d'accident causé par la prise en compe de délais

celui-ci n'a aucun moyen de deviner les calculs visés. Le simulateur peut alors tenter de le faire quand même, s'arrêter, voire même essayer de deviner quand même les intentions du concepteur en utilisant des méthodes de prédiction.

Dans l'exemple de la figure 1.2, on cherche à montrer que la simulation du circuit de la figure 1.2(a) peut conduire à une faute, même avec l'hypothèse de délais constants. La valeur du signal X (*resp.* Y) est définie comme égale à celle du signal A (*resp.* B). La seule différence est que toute modification de la valeur du signal A prend trois fois plus de temps à être répercutée sur le signal X que pour B. On peut voir pour l'exemple de la figure 1.2(b) que la valeur de Z connaît un artefact dû au fait que l'ordre des changements (B puis A) a été pris en compte dans l'autre sens.



(a) Circuit

Porte	Liste
G_1	$\{v_1, v_2\}$
G_2	$\{v_2, v_3\}$
G_3	$\{v_4, v_5\}$
G_4	$\{v_5, v_6\}$

(b) Listes de sensibilité (signaux des entrées des portes)

Temps	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	Portes à évaluer
Init	0	0	0	0	0	0	0	0	$\emptyset$
0	1	1	0	0	0	0	0	0	G_1, G_2
1	1	1	0	1	1	0	0	0	G_3, G_4
2	1	1	0	1	1	0	1	0	$\emptyset$

(c) Etat des signaux lors de la simulation

Fig. 1.3 – Exemple de simulation événementielle d'un circuit

1.2.2 Simulation événementielle

L'activité des composants d'un système peut être importante, même dans le laps de temps d'un cycle d'horloge. L'idée de la simulation événementielle est d'utiliser comme unité de temps des fractions de délais de portes. Le nombre de pas de simulation pour un cycle est donc beaucoup plus important, mais en revanche le nombre de signaux modifiés à chaque étape est potentiellement faible, car on n'évalue que les signaux dont les entrées ont changé depuis la dernière étape. L'ordonnancement de la simulation des différents composants du système est alors dynamique, dans le sens où il est déterminé lors de la simulation. Un exemple de simulation événementielle pour laquelle on a pris un pas de temps correspondant au temps de traversée d'une porte est montré en figure 1.3.

Le principal avantage de la simulation événementielle est sa flexibilité, car elle permet de gérer indifféremment des systèmes synchrones et asynchrones avec des délais arbitraires.

Algorithme

Pour simuler le comportement d'un système par une méthode événementielle, il est nécessaire de maintenir pour chaque porte une liste des signaux pour lesquels tout événement nécessite une nouvelle évaluation, c'est à dire les signaux d'entrées. Cette liste est appelée *liste de sensibilité*. L'algorithme de simulation se découpe alors en deux étapes. Premièrement, on évalue les portes qui doivent l'être en fonction des événements sur les signaux, et toute modification d'un signal est postée dans un échéancier. Ensuite, on met à jour les signaux à partir de l'échéancier, et on détecte les événements qui sont causés par ces modifications.

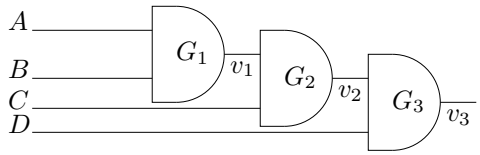
Problèmes

L'inconvénient principal de la simulation événementielle est sa lenteur, car la gestion dynamique des événements lors de la simulation impose de grands délais. On peut d'ailleurs remarquer que le surcoût induit par la gestion de l'échéancier et des listes de sensibilité est d'autant plus importante que les calculs réalisés par les portes sont simples. En effet, on pourrait considérer que, dans ce cas, la perte de temps causée par des calculs redondants (c'est-à-dire de portes dont les entrées n'ont pas été modifiées) serait compensée par le gain sur les gestions. C'est ce que fait la simulation au cycle près, présentée par la suite. De plus, on peut considérer que le graphe du système est interprété plutôt que compilé, ce qui fait que le temps de simulation est fonction de l'activité du système plutôt que de sa taille. Ceci explique le succés de cette méthode de simulation lors des années 70-80, durant lesquelles cette activité était comprise entre 1% et 20% ([Ulr69]). Cependant, la prédominance des systèmes synchrones dans les années 90, associée à une importante augmentation de la densité d'événements par cycle causée par l'émergence des techniques de parallélisme et de pipelining, a favorisé à nouveau la simulation au cycle près.

Un autre phénomène peut réduire grandement l'efficacité de la simulation événementielle. En effet, alors qu'il est primordial de n'évaluer que les portes dont l'évaluation est nécessaire, des cas très simples peuvent conduire à des événements inutiles [Hom01]. Par exemple, dans la figure 1.4, on peut voir que les évaluations de G_2 et G_3 au temps 0 sont inutiles. Une méthode pour résoudre les problèmes du type de la figure est d'utiliser des méthodes d'ordonnancement [WM90]. Dans l'exemple, l'évaluation de G_1, puis G_2, puis G_3 donne le résultat avec un nombre d'itérations optimal.

1.2.3 Simulation au cycle près

Aussi appelée simulation par code compilé, la simulation au cycle près sert à vérifier que les sorties du système sont correctes. En effet, dans un tel simulateur, la structure



(a) Circuit

Temps	A	B	C	D	v_1	v_2	v_3	Portes à évaluer
Init	0	0	0	0	0	0	0	$\emptyset$
0	1	1	1	1	0	0	0	$G_1G_2G_3$
1	1	1	1	1	1	0	0	G_2
2	1	1	1	1	1	1	0	G_3
3	1	1	1	1	1	1	1	$\emptyset$

(b) Etat des signaux lors de la simulation

Fig. 1.4 – Exemple de circuit générant des événements inutiles

du système simulé est stockée en mémoire, et chaque composant/élément logique a son propre code de simulation. La simulation au cycle près est appelée ainsi car chaque signal est évalué une fois par cycle d'horloge. Son grand avantage est son potentiel à fournir des simulations très rapides car elle élimine tous les délais induits d'une part par l'ordonnancement dynamique et d'autre part par la propagation des événements. La simulation au cycle près est parfaitement adaptée aux systèmes synchrones, cependant, comme on le verra en 1.2.4, certains simulateurs au cycle près peuvent gérer des systèmes munis de délais arbitraires.

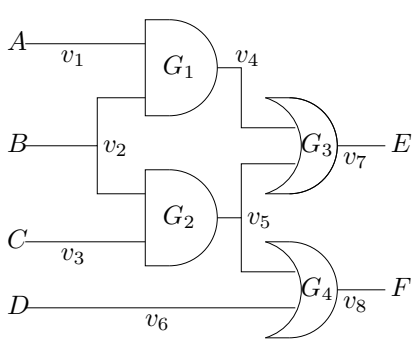
Algorithme

Si on veut simuler un système à partir d'une description de celui-ci comme un ensemble de portes logiques (figure 1.5(a)), le fonctionnement de la simulation au cycle près repose dans un premier temps sur la construction de son graphe de causalité (figure 1.5(b)).

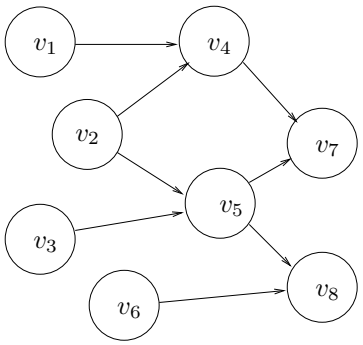
Définition 1 (Graphe de causalité d'un système). *Etant donné un système $\mathcal{S}$ décrit comme un ensemble de portes logiques et de signaux correspondant à leurs entrées et sorties respectives, on peut construire le graphe de causalité $G = (V, A)$ (aussi appelé graphe de précédence) de la façon suivante :*

(i) A chaque signal s de $\mathcal{S}$ on associe par une bijection f un sommet $v = f(s)$,

(ii) Pour tout couple de sommets de G $\{u, v\} \in V \times V$, on ajoute l'arc (u, v) à A si et seulement s'il existe une porte p dans $\mathcal{S}$ telle que le signal $f^{-1}(u)$ est une entrée de p et que le signal $f^{-1}(v)$ en est une sortie.



(a) Circuit



(b) Graphe de causalité

i	1	2	3	4	5	6	7	8
$\varphi(i)$	v_1	v_2	v_3	v_6	v_4	v_5	v_7	v_8

(c) Numérotation possible

$1 : V_1 \leftarrow A$
$2 : V_2 \leftarrow B$
$3 : V_3 \leftarrow C$
$4 : V_6 \leftarrow D$
$5 : V_4 \leftarrow \mathrm{AND}(V_1, V_2)$
$6 : V_5 \leftarrow \mathrm{AND}(V_2, V_3)$
$7 : V_7 \leftarrow \mathrm{OR}(V_4, V_5)$
$8 : V_8 \leftarrow \mathrm{OR}(V_5, V_6)$

(d) Code de simulation

Fig. 1.5 – Génération du code pour la simulation au cycle près d'un circuit

Dans un second temps, on réalise un tri topologique du graphe de causalité du système (figure 3.20(a)). Cette opération, appelée *levelisation*, correspond en fait à une numérotation du graphe (chapitre 2), et sert à s'assurer que le simulateur respecte le principe de causalité, c'est-à-dire que les sorties d'une porte ne peuvent être calculées avant ses entrées. Pour l'exemple, une numérotation possible est donnée en figure 3.20(a). Dans un second temps, une équation est générée pour chaque porte pour représenter la relation entre ses sorties et ses entrées à partir de relations logiques telles que AND, OR, etc. Finalement, le programme de simulation est généré (figure 1.5(d)).

Souvent, et puisque les 3 ensembles correspondants sont isomorphes deux à deux, on désignera de la même façon :

- Un signal du système,
- Un sommet du graphe,
- Une variable du code de simulation.

Problèmes

Les méthodes de simulation au cycle près souffrent principalement de deux inconvénients majeurs. Tout d'abord, elles sont incapables de gérer simplement les systèmes asynchrones. Ensuite, elles n'offrent pas d'informations temporelles et de délais dans leur modèles, ce qui ne leur permet pas d'être appliquées telles quelles dans beaucoup de cas (voir section 1.2).

Un autre inconvénient de la simulation au cycle près est venu de la croissance exponentielle de la taille des sytèmes considérés par les concepteurs. En effet, puisque chaque porte du système est évaluée lors de chaque cycle, les temps de simulation peuvent devenir rédhibitoires même si l'activité globale du système est faible.

De plus, la simulation au cycle près, à la différence de la simulation événementielle, assume que le système simulé est synchrone, c'est-à-dire que l'activité de tous les éléments de celui-ci est synchronisée sur une horloge commune. En pratique, presque tous les systèmes ont justement des connections asynchrones [Jen91, DeV97]... En réalité, cette situation a tendance à évoluer dans le temps. Par exemple, au début des année 90, les circuits étaient majoritairement synchrones [SS92] alors qu'actuellement la tendance est aux systèmes localement synchrones mais asynchrones dans leur ensemble.

L'inconvénient final de la simulation au cycle près est celui sur lequel portent nos travaux. Ce point part du constat que le code de simulation est compilé à l'aide d'un compilateur classique, et que les options d'optimisation habituellement rencontrées sur de tels compilateurs ne sont pas adaptées aux codes générés pour la simulation. En effet, ils ne comportent ni boucles ni instructions de branchement, éléments qui sont prépondérants dans les fameuses options de compilation. Or, devant la croissance de la taille des systèmes à simuler, les problèmes sont de deux natures. Tout d'abord, l'utilisation des options d'optimisation du compilateur est à proscrire car elles font appel à des algorithmes extrêmement coûteux en temps pour un résultat nul.

Ensuite, comme le montre le tableau 1.1 dans le cas où l'on permute les évaluations des signaux V_4 et V_5, tous les ordres topologiques possiblement induits par le graphe de causalité d'un système mènent à un code de simulation différent. Or, à l'échelle d'un système, différents ordres peuvent conduire à des durées de simulation très différentes (chapitre 4). Ceci est dû à la gestion de la mémoire lors de l'exécution (défauts de cache). C'est sur ce point que portent les travaux de cette thèse. En proposant des modèles simples de représentation de la mémoire (chapitre 2), nous proposons des méthodes permettant de conduire à un ordre d'activation des portes lors de la simulation permettant de minimiser les défauts de cache. La simplicité des modèles, tout en représentant de façon réaliste les états de la mémoire comme le montrent nos résultats pratiques (chapitre 4), permet de ne pas obtenir des algorithmes lourds en calculs, ce qui ne conviendrait pas à la taille des systèmes simulés.

$1 : V_1 \leftarrow A$	$1 : V_1 \leftarrow A$
$2 : V_2 \leftarrow B$	$2 : V_2 \leftarrow B$
$3 : V_3 \leftarrow C$	$3 : V_3 \leftarrow C$
$4 : V_6 \leftarrow D$	$4 : V_6 \leftarrow D$
$5 : V_4 \leftarrow \text{AND}(V_1, V_2)$	$5 : V_5 \leftarrow \text{AND}(V_2, V_3)$
$6 : V_5 \leftarrow \text{AND}(V_2, V_3)$	$6 : V_4 \leftarrow \text{AND}(V_1, V_2)$
$7 : V_7 \leftarrow \text{OR}(V_4, V_5)$	$7 : V_7 \leftarrow \text{OR}(V_4, V_5)$
$8 : V_8 \leftarrow \text{OR}(V_5, V_6)$	$8 : V_8 \leftarrow \text{OR}(V_5, V_6)$

Tab. 1.1 – Codes différents pour la simulation du circuit de la figure 1.5

1.2.4 Etat de l'art sur la simulation au cycle près

L'objectif de cette section est de présenter comment les concepteurs de simulateurs au cycle près gèrent les différents problèmes décrits dans la section précédente. Par exemple, comment on peut simuler les parties localement synchrones d'un système au cycle près, et simuler le comportement de ces différentes parties à un plus haut niveau d'abstraction à l'aide de méthodes événementielles. Dans [SS92], une méthode est proposée pour pourvoir à l'incapacité de la compilation au cycle près de simuler les propagations de délais. Cette méthode repose sur une algèbre sur l'ensemble des signaux vus comme des ondes. D'autres méthodes utilisent des réseaux de Petri. Une méthode intéressante proposée dans ([RD05]) utilise une classe de réseau de Petri appelée réseaux de Petri colorés restreints, permettant d'appliquer des méthodes existantes en limitant l'explosion combinatoire. L'utilisation de BDD (Binary Decision Diagrams) à la fin de chaque étape pour déterminer l'état suivant ([UMR99]) est aussi rencontrée dans la littérature. Avec la connaissance à l'avance de tous les délais des portes, [FLLO95] propose une méthode permettant de compiler un simulateur événementiel en utilisant des graphes d'événements.

On peut aussi citer des méthodes utilisant la co-simulation matérielle/logicielle ([SBR05, Row94]). Dans de tels systèmes, un coprocesseur destiné exlusivement à la simulation est implanté. Selon l'objectif de la simulation, un compromis est réalisé entre la vitesse et la précision en simulant à l'aide d'un logiciel certaines parties.

1.3 Conclusion

Nos travaux portent sur un réarrangement d'un code de simulation obtenu avec une méthode quelconque. Ce réarrangement vise à accélérer l'exécution du code compilé par la minimisation d'une fonction de coût reposant sur des modèles de mémoire décrits dans le chapitre 2. Il est important d'insister sur le fait que notre méthode intervient après la génération du code par un simulateur au cycle près, lors d'une phase de pré-compilation.

Les principales contraintes que nous impose le problème sont au nombre de deux. Tout d'abord, il est important que le temps d'application de notre méthode ne compense pas le gain à l'exécution. En d'autres termes, si les modèles de mémoire que nous proposons sont trop complexes, et devant la taille des systèmes dont nous voulons accélérer la simulation, la phase de précompilation sera trop longue, et les gains lors de l'exécution du simulateur seront trop faibles pour justifier son utilisation. Cependant, nos modèles doivent rester réalistes. En effet, si les réarrangements de codes que nous proposons reposent sur des modèles de mémoire trop éloignés de la réalité, les optimisations risquent de porter sur une vision biaisée de la machine de simulation.

Les modèles que nous montrons dans le chapitre suivant sont suffisamment simples pour conduire à des résultats théoriques intéressants, et pour être utilisables en pratique, mais nous montrons de façon expérimentale (chapitre 4) que les codes obtenus par l'utilisation d'heuristiques adaptées mènent à des simulations plus rapides.

Chapitre 2

Modèles de mémoire

Comme nous l'avons vu dans le chapitre précédent, un simulateur au cycle près n'est autre qu'un programme fonctionnant sur un ordinateur, dont on veut qu'il fonctionne le plus vite possible en respectant les objectifs de simulation. Or, comme il est montré dans ce chapitre, l'utilisation des ressources de stockage des données est un enjeu crucial pour la rapidité de l'exécution de tâches sur un ordinateur. Nous présentons donc d'abord l'organisation de la mémoire d'un ordinateur et la manière dont elle est gérée lors de l'exécution d'un programme. Dans un second temps, nous montrons deux fonctions de coût destinées à donner une estimation relative, à partir du graphe de précédence d'un système, du temps d'exécution du programme compilé, sur la base des accès aux variables réalisés lors de l'exécution. Ces deux modèles servent à représenter de façon simple le fait que lorsqu'une information n'est pas utilisée depuis longtemps, son exploitation est plus difficile car les mécanismes de gestion de mémoire de l'ordinateur sont souvent conçus pour favoriser les données les plus fréquemment ou les plus récemment utilisées. Nous conclurons en présentant un résultat théorique permettant d'établir de façon simple la non ambiguïté de la seconde fonction de coût, ainsi qu'une observation empirique sur la corrélation linéaire entre les deux critères.

2.1 Mémoire d'un ordinateur

La rapidité d'un programme fonctionnant sur un ordinateur n'est pas liée exclusivement aux calculs. En effet, les accès à la mémoire qui sont réalisés pour la récupération des données peuvent donner lieu à des durées non négligeables. Ceci vient du fait que la mémoire est organisée en différents niveaux, et que les temps d'accès à ces niveaux ne sont pas identiques. On peut dire qu'on franchit un ordre de grandeur en taille et en durée d'accès à chaque fois qu'on change de niveau. Tout d'abord, nous montrons dans cette section comment se présentent les différents niveaux hiérarchiques de la mémoire. Ensuite, nous présentons succinctement différentes politiques de gestion.

Niveau	Vitesse(cycles)	Taille(octets)
Registre	1	10^2
Cache de niveau 1	10	10^4
Cache de niveau 2	10^2	10^5
Cache de niveau 3	$> 10^2$	10^6
Mémoire principale	10^3	10^9
Stockage	10^5	∞

Tab. 2.1 – Description des différents niveaux de mémoire habituels

2.1.1 Hiérarchie de la mémoire

Un ordinateur classique dispose de différents niveaux de stockage. Chacun de ces niveaux peut être représenté par trois paramètres importants : son coût, sa taille, et sa vitesse. La vitesse, représentant la durée nécessaire à l'accès à une information, est souvent mesurée en cycles du processeur. L'ensemble constitue une hiérarchie de stockage, dans laquelle les niveaux les plus proches du processeurs sont les plus rapides, les plus petits, et les plus chers (tableau 2.1 d'après [Han93]). Le niveau le plus rapide est constitué par les registres, qui servent en pratique à stocker l'information en cours de traitement. A l'inverse, les niveaux les plus lents sont de capacité très grande (considérée comme infinie en pratique), et peuvent prendre différentes formes. En général, ce sont des disques durs, mais on peut aussi stocker des données sur des disquettes, clés USB, bandes magnétiques, etc. On peut même augmenter encore les temps d'accès et la capacité en considérant les réseaux de stockage.

2.1.2 Gestion de la mémoire

La notion de *gestion de la mémoire* désigne la façon de coordonner et contrôler l'utilisation de la mémoire dans un ordinateur. Elle peut être séparée en trois notions.

Matériel Regroupe tous les composants électroniques (et les circuits associés) qui sont destinés au stockage des états d'un ordinateur (RAM, ROM, disques...).

Système d'exploitation Le système d'exploitation utilise les moyens matériels pour allouer de l'espace mémoire de la hiérarchie de stockage à ses différentes activités. Il s'occupe aussi de sécurité et de protection de la mémoire, en s'assurant son intégrité devant les erreurs intentionelles (attaque) ou non (erreur de programmation).

Application Pour optimiser leur fonctionnement (vitesse d'exécution, espace, etc...), les applications s'appuient entre autre sur une gestion optimisée de la mémoire. Par exemple, les compilateurs sont dotés d'options de compilation permettant de placer les données d'un programme au mieux lors de l'exécution.

2.1.3 La mémoire cache

Définition La *mémoire cache* est une mémoire physique de petite taille et très rapide (tableau 2.1), qui est utilisée pour stocker des copies de parties de la mémoire principale lors de l'exécution d'un programme de façon à y avoir accès rapidement. La plupart des systèmes des ordinateurs de bureau possèdent en fait plusieurs niveaux de cache –3 en général–, de plus en plus grand et de moins en moins rapide quand leur nombre augmente. La partie de la mémoire principale stockée, et la façon dont elle est stockée peuvent être multiples, et ce choix dépend de la *politique de cache*. Il ne faut pas confondre la mémoire cache avec un autre type de cache au sens plus large, dont elle est un cas particulier. En effet, on appelle aussi "cache" toute partie d'une mémoire allouée au stockage d'une partie d'une mémoire plus lente.

Politiques de cache Toute cache est dotée d'une politique de gestion destinée à tenter de prévoir le futur de façon à répondre de la meilleure manière possible aux requêtes, c'est-à-dire le plus vite possible. Les politiques de cache peuvent être implémentées en matériel, en logiciel, voire les deux simultanément. Certains systèmes autorisent même les programmes à influencer leur politique de cache, toujours dans un but d'optimisation. Trois aspects du comportement d'une cache sont contrôlés par la politique de cache :

- Quelles données sont ramenées en cache lors d'une requête pour une information non cachée, c'est-à-dire non disponible directement ? En effet, lorsque de l'information est chargée, il est courant de charger en même temps une partie de l'information voisine physiquement, en supposant de fait que celle-ci sera peut-être utilisée prochainement. Ceci est particulièrement valable pour le traitement du signal par exemple, où l'on manipule des vecteurs et des matrices de données.
- Quelles données sont supprimées de la cache en cas de manque d'espace ?
- Quand et comment sont synchronisées les modifications de données dans le cache avec le stockage sous-jacent ?

Comment les politiques de cache tentent-elles de répondre à des questions avant même qu'elles ne soient posées ? Pour le choix de l'information à supprimer en cas de manque d'espace, différentes heuristiques existent, parmi lesquelles :

Least Recently Used (LRU) (*utilisée le moins récemment*) Cette méthode conjecture que l'information qui n'a pas été utilisée depuis le plus de temps est la moins susceptible d'être utilisée à nouveau,

Least Frequently Used (LFU) (*utilisée le moins fréquemment*) Cette méthode garde trace des fréquences d'utilisation, et considère que les chargements trop fréquents de variables non disponibles sont trop coûteux.

Coût Une méthode possible est aussi de garder les éléments dont le coût de chargement est trop élevé.

Péremption Certains types de cache stockent tout simplement des données périssables. C'est par exemple le cas pour la mémoire d'un navigateur web.

Il est important de remarquer que les politiques de cache efficaces reposent sur un compromis important entre efficacité et complexité, car le temps gagné au niveau des chargements de données ne doit pas être dépassé par celui passé à la mise en application de la politique.

2.1.4 Cas particulier de la simulation de systèmes synchrones

Les simulateurs au cycle près de systèmes synchrones sont compilés à l'aide de compilateurs classiques. Or, les options de compilation utilisées pour optimiser la gestion de la mémoire lors de l'exécution d'un programme sont destinés aux programmes structurés, par exemple avec les boucles ou les nids de boucles (voir par exemple [Pug91]). Le problème est que les programmes de simulation ne possèdent ni boucles ni instructions de branchement succeptibles d'être affectées par de telles optimisations. La conséquence est double : les optimisations ont une durée rédhibitoire, et n'apportent aucune amélioration.

L'approche que nous avons choisi de présenter dans le chapitre suivant est de reposer sur une modélisation très simple de la mémoire lors de l'exécution d'un programme. Cette modélisation nous mène ensuite à donner une estimation du coût d'un code source en terme de fréquence d'utilisation des variables ou de leur durée de vie, sur le modèle des politiques prédictives de cache. Ensuite, ces deux modèles donnent lieu (chapitre 3) à une étude théorique poussée. Dans le chapitre 4, nous vérifions expérimentalement que nos modèles représentent une vue adéquate de la réalité, malgré leur simplicité.

2.2 Modèles

L'objectif de cette section est de présenter de façon détaillée deux modèles mathématiques qui représentent une estimation du coût des accès à la mémoire lors de l'exécution d'un programme, représenté par son graphe de causalité. Dans un premier temps, nous nous attacherons à définir les objets mathématiques en jeu, ainsi que les notations et conventions utilisées. Ensuite, nous présenterons le modèle UCS (Uniform Cost Stack), puis le modèle DSC (Directed SumCut).

2.2.1 Généralités

Notations et définitions

Dans tout ce chapitre, $G = (V, A)$ est un graphe orienté sans circuit. Dans le cadre de nos travaux, G est le graphe de causalité des variables d'un code de simulation. On note

$n = |V|$ le nombre de sommets de G, et $m = |A|$ le nombre d'arcs de G. Pour chaque sommet $u \in V$, $\Gamma^+(u)$ (*resp.* $\Gamma^-(u)$) est l'ensemble des successeurs (*resp.* prédécesseurs) de u. On note finalement $\delta^+(u) = |\Gamma^+(u)|$ et $\delta^-(u) = |\Gamma^-(u)|$ les degrés respectivement sortants et entrants de u.

Définition 2 (Numérotation d'un graphe). *Soit $\varphi : V \to \{1, \cdots, n\}$. Alors φ est une* numérotation de G *si :*
- *φ est une bijection,*
- *φ vérifie la contrainte de précédence, c'est-à-dire que pour tout arc $a = (u, v) \in A$, on a $\varphi(u) < \varphi(v)$.*

Pour tout $i \in \{1, \cdots, n\}$, la notation $\varphi^{-1}(i)$ est souvent utilisée pour désigner le sommet $v \in V$ dont le numéro dans φ est i. On note finalement Φ_G l'ensemble des numérotations possibles de G, qu'on notera Φ pour des raisons de simplicité quand le contexte le permettra.

Fonction de coût

Soit une numérotation φ de G et un arc $a = (u, v) \in A$. Si l'on considère que G est le graphe de causalité d'un système, lors de l'exécution du code de simulation correspondant, le calcul de la variable associée au signal v nécessitera un chargement de la valeur associée au signal u. Le coût de ce chargement est noté $\mathcal{C}(\varphi, (u, v))$. Cette fonction dépendra du modèle de mémoire considéré (sous-sections 2.2.3 et 2.2.4). Puisque tous les accès à la mémoire sont effectués de manière séquentielle, le coût total de la numérotation φ pour le sommet u sera simplement la somme des coûts de ses arcs sortants, c'est-à-dire la somme de tous les accès à la variable associée au signal u :

$$\mathcal{C}(\varphi, u) = \sum_{v \in \Gamma^+(u)} \mathcal{C}(\varphi, (u, v))$$

De la même manière, le coût total associé au graphe G pour la numérotation φ sera :

$$\mathcal{C}(\varphi, G) = \sum_{u \in V} \mathcal{C}(\varphi, u)$$

Il est intéressant de remarquer que puisque nous considérons que les temps de traversée des portes logiques sont constants, il n'est pas nécessaire de les inclure dans notre fonction de coût puisque la somme de tous les délais ne dépendra pas de la numérotation. Dans la suite, après avoir présenté un état de l'art sur les différents problèmes de numérotation existants, nous présentons deux modèles simples de gestion des accès à la mémoire sous la forme de fonctions de coût déterministes d'une part, et ne dépendant que de φ d'autre part. Dans le premier modèle, l'estimation de $\mathcal{C}(\varphi, u)$ prend en compte le nombre d'instructions effectuées entre deux utilisations successives de la variable u, alors que le second modèle présenté est plus complexe car il mémorise l'ensemble des opérations de lecture et d'écriture en mémoire.

2.2.2 Problèmes de numérotation de graphes

Le cadre dans lequel se placent nos travaux est celui des problèmes de numérotation de graphes. La référence dans ce domaine est l'inventaire réalisé par Diaz *et al.* [DPS02], duquel les exemples de cette partie sont inspirés. Cependant, l'immense majorité des résultats concernent les problèmes non orientés. L'objectif de cette section est de présenter les principaux problèmes dans leurs versions orientée et non orientée et les notions qui leur sont associées, ainsi que les résultats correspondants.

Problèmes non orientés

Dans cette partie, on considère un graphe non orienté $G = (V, E)$, et on pose $n = |V|$. Dans le cas de graphes non orientés, la notion de numérotation est relâchée, car la contrainte de précédence n'est plus applicable. Les numérotations de graphes non orientés sont rencontrées sous plusieurs dénominations (linear arrangement, linear ordering, numbering, labeling,...), mais représentent toujours une application bijective $\varphi : V \mapsto \{1, \cdots, n\}$. Afin de présenter les principaux problèmes, plusieurs définitions sont nécessaires.

Etant donnés une numérotation φ de G et un entier i dans $\{1, \cdots, n\}$, on définit les deux ensembles suivants :

$$\begin{aligned} L(i, \varphi, G) &:= \{u \in V, \varphi(u) \leq i\} \\ R(i, \varphi, G) &:= \{u \in V, \varphi(u) > i\} \end{aligned}$$

La *coupe arête* (edge cut) à la position i de φ est définie comme :

$$\theta(i, \varphi, G) := |\{\{u, v\} \in E, u \in L(i, \varphi, G) \text{ et } v \in R(i, \varphi, G)\}|$$

La *coupe arête modifiée* (modified edge cut) à la position i de φ est définie comme :

$$\zeta(i, \varphi, G) := |\{\{u, v\} \in E, u \in L(i, \varphi, G) \text{ et } v \in R(i, \varphi, G) \text{ et } \varphi(u) \neq i\}|$$

La *coupe sommet* (vertex cut, ou séparation) à la position i de φ est définie comme :

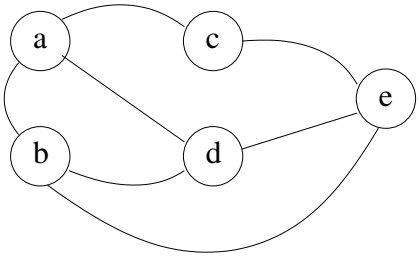
$$\delta(i, \varphi, G) := |\{u \in L(i, \varphi, G), \exists v \in R(i, \varphi, G) \text{ et } \{u, v\} \in E\}|$$

On définit finalement pour toute arête $\{u, v\} \in E$ sa *longueur* :

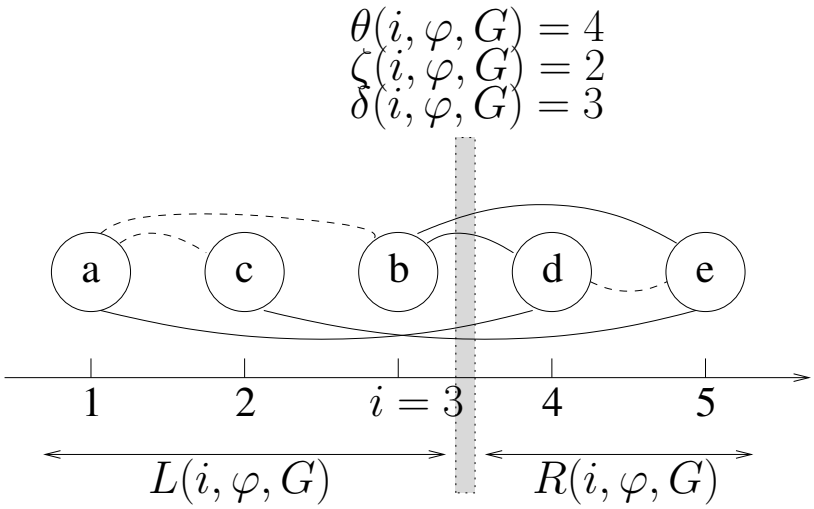
$$\lambda(\{u, v\}, \varphi, G) := |\varphi(u) - \varphi(v)|$$

$L(i,\varphi,G)$	$=$	$\{u \in V, \varphi(u) \leq i\}$
$R(i,\varphi,G)$	$=$	$\{u \in V, \varphi(u) > i\}$
$\theta(i,\varphi,G)$	$=$	$\lvert\{\{u,v\} \in E, u \in L(i,\varphi,G) \text{ et } v \in R(i,\varphi,G)\}\rvert$
$\zeta(i,\varphi,G)$	$=$	$\lvert\{\{u,v\} \in E, u \in L(i,\varphi,G) \text{ et } v \in R(i,\varphi,G) \text{ et } \varphi(u) \neq i\}\rvert$
$\delta(i,\varphi,G)$	$=$	$\lvert\{u \in L(i,\varphi,G), \exists v \in R(i,\varphi,G) \text{ et } \{u,v\} \in E\}\rvert$
$\lambda(\{u,v\},\varphi,G)$	$=$	$\lvert\varphi(u) - \varphi(v)\rvert$

TAB. 2.2 – Mesures pour les numérotations de graphes



(a) Exemple de graphe non orienté



(b) Représentation linéaire du graphe

FIG. 2.1 – Différentes mesures pour les numérotations de graphes

Ces mesures sont résumées dans le tableau 2.2 pour plus de clarté.

La représentation la plus répandue pour représenter une numérotation φ d'un graphe G consiste à placer les sommets de G sur une ligne horizontale en correspondance avec leur numéro, comme montré en figure 2.1(b). Cette représentation graphique permet d'illustrer simplement les définitions précédentes. Par exemple, en dessinant une ligne verticale juste après la position i et avant la position $i+1$, les sommets à gauche de la ligne sont ceux de $L(i, \varphi, G)$, et ceux à droite sont ceux de $R(i, \varphi, G)$. Le calcul de la coupe arête $\theta(i, \varphi, G)$ se fait en comptant les arêtes qui coupent la ligne verticale. En ne comptant pas les arêtes adjacentes au sommet $\varphi^{-1}(i)$, on obtient la coupe arête modifiée $\zeta(i, \varphi, G)$. Pour calculer la coupe sommet $\delta(i, \varphi, G)$, il suffit de compter les sommets à gauche de la ligne qui sont reliés à un sommet à droite de la ligne. Finalement, la calcul de la longueur $\lambda(\{u, v\}, \varphi, G)$ consiste simplement à mesurer la longueur entre les sommets u et v.

Les fonctions de coût ainsi définies donnent lieu à une série de problèmes de décision, parmi lesquels (de façon non exhaustive) :

Problème: Vertex Separation

Instance: $G = (V, E)$ un graphe non orienté, un entier K.

Question: Est-il possible de trouver une numérotation $\varphi : V \to \{1, ..., |V|\}$ telle que

$$\max_{i \in \{1, \cdots, n\}} \delta(i, \varphi, G) \leq K$$

Problème: Sum Cut

Instance: $G = (V, E)$ un graphe non orienté, un entier K.

Question: Est-il possible de trouver une numérotation $\varphi : V \to \{1, ..., |V|\}$ telle que

$$\sum_{i \in \{1, \cdots, n\}} \delta(i, \varphi, G) \leq K$$

Problème: Profile

Instance: $G = (V, E)$ un graphe non orienté, un entier K.

Question: Est-il possible de trouver une numérotation $\varphi : V \to \{1, ..., |V|\}$ telle que

$$\sum_{u \in V} (\varphi(u) - \min_{\{u,v\} \in E} \varphi(v)) \leq K$$

Problème: Minimum Linear Arrangement (minLA)

Instance: $G = (V, E)$ un graphe non orienté, et un entier K.

Question: Est-il possible de trouver une numérotation $\varphi : V \rightarrow \{1, ..., |V|\}$ telle que

$$\sum_{\{u,v\} \in E} \lambda(\{u, v\}, \varphi, G) \leq K$$

Tous ces problèmes sont NP-complets en général([DPS02]), mais peuvent être résolus en temps polynomial pour certaines classes de graphes.

Le problème Sum Cut, qui nous intéresse plus particulièrement, a été introduit par [DGPT91] comme une version simplifiée du problème du δ-opérateur [Dia79]. Le problème Profile a été présenté par [LY94] pour optimiser le stockage de certaines matrices creuses. Il se trouve que les deux sont en fait équivalents au problème de complétion des graphes de comparabilité [RAK91], qui a des applications en archéologie [Ken69] et en bioinformatique, pour la comparaison de fragments d'ADN [Kar93]. De manière plus générale, les problèmes de placement et de routage lors de la conception de systèmes intégrés peuvent être modélisés par des numérotations de graphes [ST97].

Problèmes orientés

En rajoutant la contrainte dite *de précédence*, on peut adapter toutes les fonctions de coût et les problèmes associés aux graphes orientés sans circuits. Plusieurs de ces problèmes de numérotation ont été étudiés dans le passé. Cependant, la version orientée du problème SumCut, présentée en détails dans la suite, n'a pas été étudiée avant nos travaux.

2.2.3 Directed Sum Cut (DSC)

La première fonction de coût que nous proposons part du constat que toute numérotation du graphe de causalié d'un système induit une numérotation des lignes du code de simulation. Par exemple, dans la figure 1.5(d), $\varphi^{-1}(V_4) = 5$ signifie que la variable correspondant au signal V_4 est créée à la ligne 5 du code. Comme elle est utilisée pour la dernière fois à la ligne 7, on dit qu'elle a une *durée de vie* de 3.

Afin de généraliser cette notion de durée de vie d'une variable, on considère une variable u juste après sa création. Le premier accès à u est effectué pour le calcul de son premier successeur correspondant dans φ, noté $s_1(\varphi, u)$, ou plus simplement $s_1(u)$. L'hypothèse

que nous faisons est que le coût de la lecture de u est proportionnel au nombre d'accès à la mémoire depuis la création de u. Nous considérons que ce nombre est fonction du nombre de lignes de code entre la création et l'utilisation de u, c'est-à-dire $\mathcal{C}(\varphi,(u,s_1(u))) = f(\varphi(s_1(u)) - \varphi(u))$, où f est une fonction croissante. Dans la suite, on supposera que $f \equiv \mathrm{Id}$. Cette hypothèse est cohérente par rapport à l'interprétation de la signification des variables manipulées, puisqu'elles sont toutes identiques.

Suite à ce calcul, les variables u et $s_1(u)$ sont supposées être placées dans des niveaux de mémoire équivalents. La conséquence de ceci est que lors de l'accès à u pour le calcul de son deuxième successeur $s_2(u)$, le coût d'accès est $\mathcal{C}(\varphi,(u,s_2(u))) = \varphi(s_2(u)) - \varphi(s_1(u))$, puisque $\varphi(s_2(u)) - \varphi(s_1(u))$ est le nombre de lignes de code depuis la création de $s_1(u)$. Par conséquent, le coût total associé à la variable u est :

$$\mathcal{C}(\varphi,u) = \begin{cases} \sum_{i=1}^{\delta^+(u)} f(\varphi(s_i(u)) - \varphi(s_{i-1}(u))) & \text{si } \delta^+(u) \geq 1, \\ 0 & \text{sinon} \end{cases}$$

où $\delta^+(u)$ est le degré sortant de u dans G, $s_0(u) = u$, et $s_1(u), s_2(u), \cdots$ sont les successeurs de u numérotés selon φ. Ce coût peut être aussi être exprimé d'une façon très simple, en ne prenant en compte que la différence entre la date de création de u et sa dernière utilisation, c'est-à-dire sa durée de vie :

$$\mathcal{C}(\varphi,u) = \left(\max_{(u,v)\in A} \varphi(v)\right) - \varphi(u) \tag{2.1}$$

Le coût total pour G sera donc calculé comme suit :

$$\mathcal{C}(\varphi) = \mathcal{C}(\varphi,G) = \sum_{u\in V} \left(\left(\max_{(u,v)\in A} \varphi(v)\right) - \varphi(u)\right).$$

Le théorème suivant montre que l'expression de cette fonction de coût peut être reliée très simplement à un critère classique de numérotation de graphe, *Directed SumCut*. La *coupe sommet* à la position i représente le nombre de variables actives, c'est-à-dire déjà créées et encore nécessaire pour des calculs ultérieurs à la position i.

En termes de gestion de la mémoire lors de l'exécution d'un programme, l'interprétation pour une variable donnée w de $w \notin \{u \in V : \varphi(u) \leq i \wedge (\exists v : \varphi(v) > i \wedge (u,v) \in A)\}$ est effectivement la suivante : soit w n'est plus utilisée, soit elle n'a pas encore été créée.

Théorème 2.2.1. *Si on définit la* Directed SumCut *comme*

$$\mathrm{DSC}(\varphi,G) = \sum_{1\leq i\leq n} \delta(i,\varphi,G),$$

alors

$$\mathcal{C}(\varphi,G) = \mathrm{DSC}(\varphi,G)$$

Preuve Pour tout couple $(u, v) \in V \times V$, on considère l'indicateur $\xi(u, v)$ valant 1 s'il existe un arc $(u, w) \in A$ tel que $\varphi(u) \leq \varphi(v) < \varphi(w)$ et égal à 0 sinon. Par définition, on a $\delta(\varphi(v), \varphi, G) = \sum_u \xi(u, v)$, donc $\mathrm{DSC}(\varphi, G) = \sum_v \sum_u \xi(u, v) = \sum_u \sum_v \xi(u, v)$. La somme intérieure est égale au nombre de sommets v qui sont numérotés dans l'intervalle $\{\varphi(u), \cdots, \max_{(u,w)\in A} \varphi(w) - 1\}$, c'est-à-dire $\left(\max_{(u,v)\in A} \varphi(v)\right) - \varphi(u)$. □

Un exemple pour le théorème est donné en figure 2.2. Dans la partie droite de la figure se trouve un exemple de code de simulation, et les points représentent les variables actives à chaque étape. Il est similaire de compter les points par colonne (coût $\mathcal{C}$) ou en lignes (coût DSC).

Dans la suite, la fonction de coût ainsi définie sera donc appelée DSC. Nous utiliserons aussi toutes les notations correspondantes : $\mathrm{DSC}(\varphi, (u, s_i(u))) = \varphi(s_i(u)) - \varphi(s_{i-1}(u))$ et $\mathrm{DSC}(\varphi, u) = \max_{(u,v)\in A} \varphi(v) - \varphi(u)$. Un exemple de calcul du coût DSC d'un graphe est donné en figure 2.3. Pour la numérotation de l'exemple, on trouve un coût de 10.

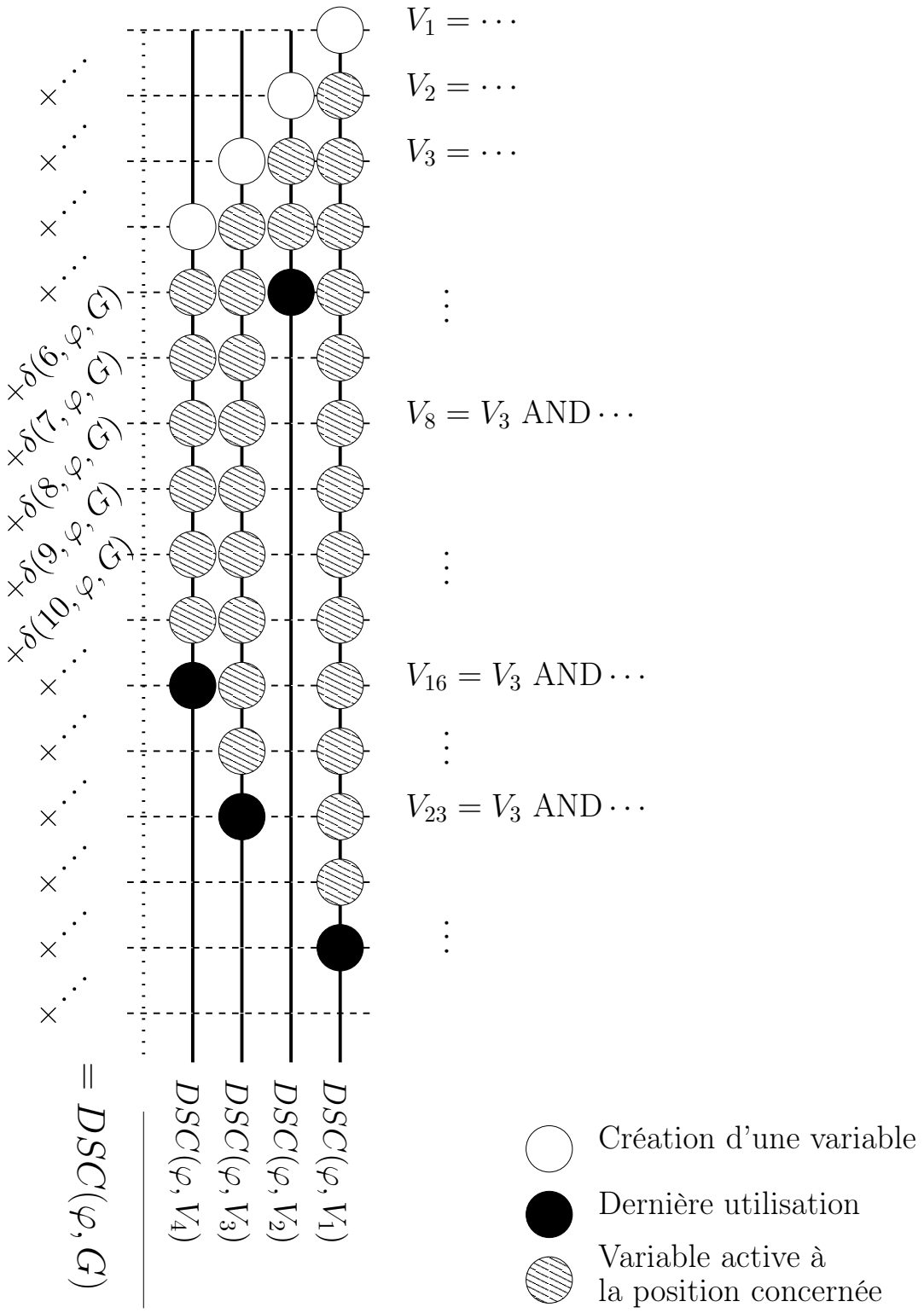
2.2.4 Uniform Cost Stack (UCS)

Le modèle UCS (pour *Uniform Cost Stack*, ou pile à coût uniforme), est destiné à représenter les coûts de toutes les variables qui sont gardées en mémoire lors de l'exécution d'un programme. On considère que la mémoire est organisée comme une pile, et que le coût d'accès à une variable est proportionnel à sa distance au sommet. Ce modèle est en fait une extension de celui de Sethi pour les problèmes d'allocation de registres [Set75, BGT98].

La mémoire est vue comme une pile, pour laquelle 3 opérateurs sont disponibles :

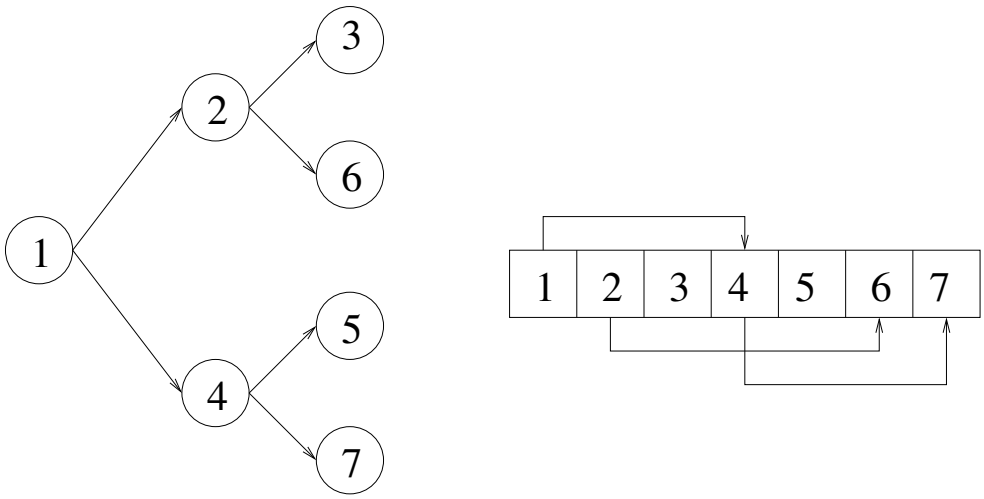
- RD(α) lit la valeur de l'entrée pour la variable α et la place au sommet de la pile. Comme on considère que la durée de cette opération est constante, dans le modèle elle est considérée comme nulle.
- LD(α) déplace la variable α à partir d'un endroit de la pile jusqu'au sommet. Le coût de cette opération est considéré comme proportionnel à la longueur du déplacement, c'est-à-dire à la distance au sommet avant le déplacement.
- OP($\alpha_1, \cdots, \alpha_k$) applique l'opérateur OP aux variables $\alpha_1, \cdots, \alpha_k$. On suppose que celles-ci ont préalablement été placées (grâce à des opérations LD) aux k niveaux de la pile les plus proches du sommet. L'ordre d'empilement n'a pas d'importance. Cette hypothèse peut être justifiée par le fait que dans la réalité l'ordre de stockage des paramètres dans les registres n'influence pas le résultat d'une opération. Le resultat de l'application de OP est placé au sommet de la pile, l'ordre respectif des paramètres restant inchangé. Comme le coût d'une telle opération est suppposé constant, on considère qu'il est nul pour le modèle.

Pour toute variable dans la pile, la distance au sommet est appelée la *profondeur*. Par exemple, on considère le graphe de la figure 2.4(a). La numérotation correspond aux éti-



(a) Exemple (b) Légende

Fig. 2.2 – Exemple pour le théorème 2.2.1



(a) Graphe avec une numérotation

	1	2	3	4	5	6	7
Coût	3	4	0	3	0	0	0

(b) Coût des sommets

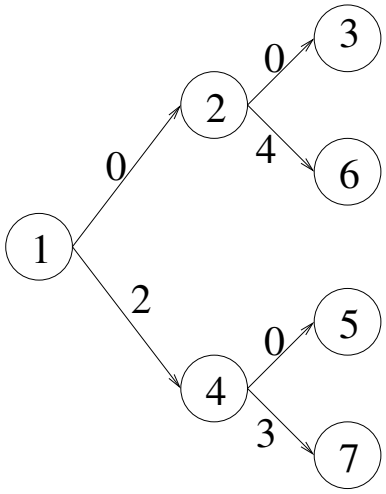
FIG. 2.3 – Exemple de calcul du coût DSC

quettes des sommets. Pour celle-ci, on peut déduire de façon simple une liste d'opérations RD, LD et OP qui sont effectuées pour l'évaluation du graphe. La figure 2.4(b) représente les différents états de la pile après ces opérations (LD(i) signifie "charger la variable $\varphi^{-1}(i)$"). Le coût total d'une exécution est la somme de tous les coûts des chargements (opérations LD) car chacun est associé à un arc du graphe. Sur la figure 2.4(a), les arcs sont valués par les coûts correspondants. Dans cet exemple, on a un coût total de 9.

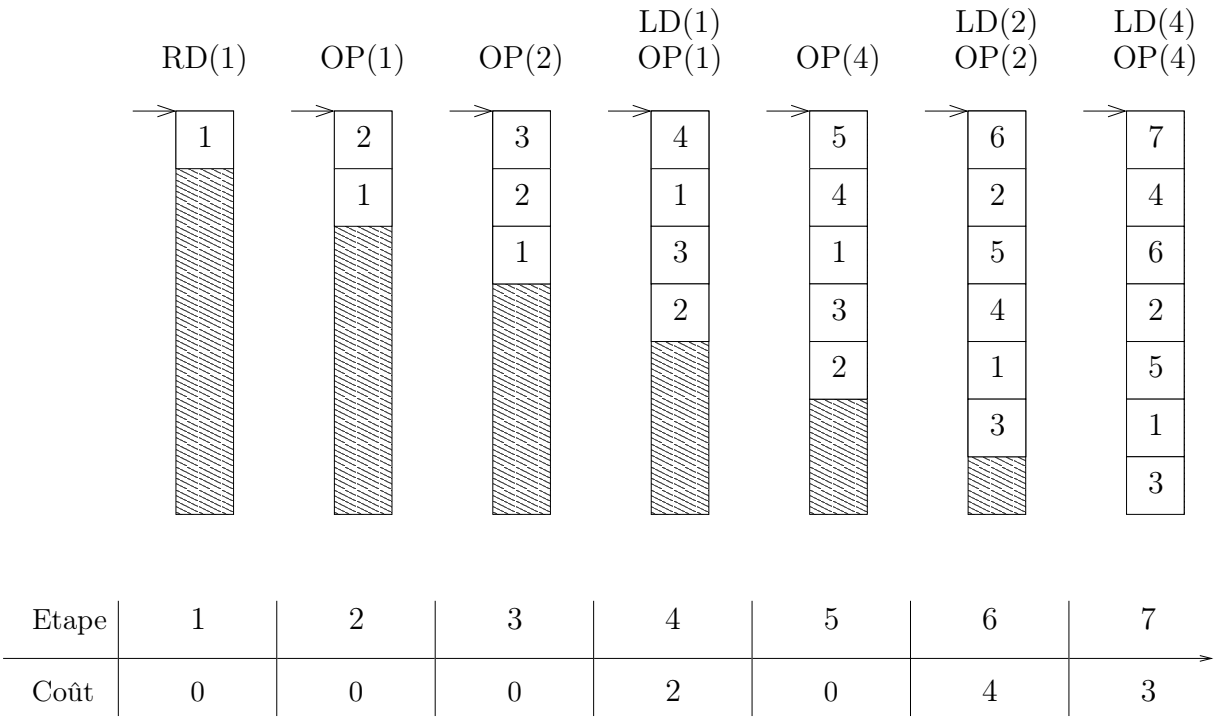
Pour le cas des graphes quelconques, la génération du code associé à une numérotation donnée est plus compliquée. En effet, si un sommet $u \in V$ a plusieurs prédécesseurs, il est nécessaire de décider dans quel ordre ils sont empilés pour le calcul optimal de u. Une méthode algorithmique polynomiale pour empiler les successeurs de u de façon optimale est montrée à la suite.

Dans la suite, la fonction de coût ainsi définie sera donc appelée UCS. Nous utiliserons aussi toutes les notations correspondantes : UCS(φ, G) pour le coût d'un graphe, UCS($\varphi, (u, v)$) pour le coût d'un arc, et UCS(φ, u) pour le coût d'un sommet.

Dans ce qui suit, nous montrons un lemme technique particulièrement important pour le modèle UCS. En effet, comme montré en section 2.2.4, le modèle UCS tel qu'il a été défini ne garantit ni l'unicité du code de simulation, ni l'unicité du coût total.



(a) Exemple d'arborescence avec coûts UCS



(b) Evolution de la pile

Fig. 2.4 – Exemple de calcul du coût UCS

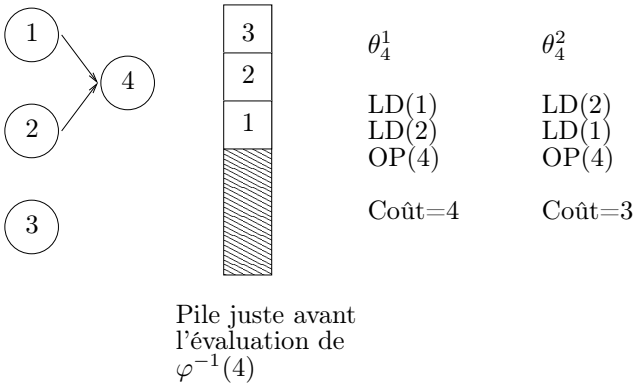


FIG. 2.5 – Deux ordres d'empilement différents pour l'évaluation de $\varphi^{-1}(4)$

Ordre optimal d'empilement pour le modèle UCS

Pour un ordre d'exécution φ donné, le modèle UCS tel qu'il a été défini ne garantit ni l'unicité du code de simulation, ni l'unicité du coût total, car un ordre doit être défini pour les opérations LD dans le cas des sommets possédant plusieurs prédécesseurs. Cet ordre est appelé *ordre d'empilement*. En effet, si on considère l'exemple de la figure 2.5, on peut constater que le coût total de l'exécution dépend de l'ordre de chargement des variables $\varphi^{-1}(1)$ et $\varphi^{-1}(2)$ pour l'évaluation de $\varphi^{-1}(4)$. Si $\varphi^{-1}(1)$ est chargée en premier, le coût est de 4, sinon il est de 3.

Dans ce qui suit, nous présentons une méthode optimale simple (c'est-à-dire algorithmique et polynomiale) qui, ajoutée au modèle UCS, permet de lever toutes les ambiguïtés sur la définition du coût.

Définition 3 (Ordre d'empilement). *Un* ordre d'empilement θ_u *d'un sommet* $u \in V$ *est une séquence finie* $(\theta_u(1), \theta_u(2), \cdots, \theta_u(K_u))$*, où* $\theta_u(i) \in \Gamma^-(u)$ *pour tout* i *tel que* $1 \leq i \leq K_u$ *(*K_u *est simplement la longueur de la séquence).* θ_u *représente la suite d'opérations* $\mathrm{LD}(\theta_u(i))$ *nécessaires pour le démarrage du calcul de* u*. L'*ordre total d'empilement *est l'ensemble* $\theta = \{(u, \theta_u), u \in V\}$*.*
Un ordre d'empilement est dit compatible *avec une numérotation* φ *si le code de simulation résultant fonctionne, c'est-à-dire si toutes les variables sont correctement chargées pour chaque opérateur* OP*. Le coût d'un ordre total d'empilement compatible* θ *pour la numérotation* φ *est noté* $\mathrm{UCS}_\theta(\varphi, G)$*, ou plus simplement* $\mathrm{UCS}_\theta(\varphi)$ *quand il n'y a pas d'ambiguïté.*

Pour l'exemple de la figure 2.5, $\theta_4^1 = (\varphi^{-1}(1), \varphi^{-1}(2))$.

La preuve de notre méthode d'empilement optimal consiste en deux lemmes. Dans un premier temps (lemme 1), nous montrons la dominance des ordres d'empilement tels que les variables qui sont au sommet de la pile, ou qui ne sont séparées de celui-ci que par des variables dont le chargement est nécessaire pour le calcul, ne sont pas chargées. Dans

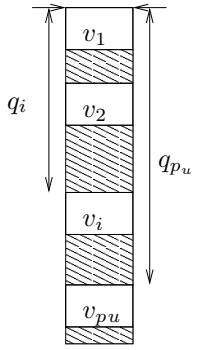


FIG. 2.6 – Distance d'un sommet au sommet de la pile

un second temps (lemme 2), nous montrons la dominance des ordres chargeant les autres variables dans l'ordre de leur distance croissante au sommet de la pile.

Soit φ une numérotation. On considère un sommet $u \in V$ sur le point d'être évalué. On note $p_u = |\Gamma^-(u)|$, et les éléments de $\Gamma^-(u)$ sont notés $v_1, \cdots, v_{p_u}$ (voir figure 2.6). On suppose qu'ils sont numérotés selon leur distance croissante au sommet de la pile, notées respectivement $q_1, \cdots, q_{p_u}$. Pour tout i dans $\{1, \cdots, K_u\}$, l'inégalité $q_i \geq i$ signifie qu'il existe une variable $v \notin \Gamma^-(u)$ entre v_i et le sommet de la pile, et que par conséquent v_i va nécessairement devoir être l'objet d'une opération LD avant que u puisse être calculé. On note i_u la valeur maximale de i parmi $\{1, \cdots, p_u\}$ vérifiant $q_i = i - 1$.

Lemme 1. *Pour toute numérotation φ, il existe un ordre d'empilement qui minimise* $\mathrm{UCS}_\theta(\varphi, G)$ *et tel que, pour tout u, $K_u = p_u - i_u$ et $\theta_u(i) \in \{v_{i_u}, \cdots, v_{p_u}\}$ (pour tout $i \in \{1, \cdots, K_u\}$) (voir figure 2.7).*

Preuve L'implication $(K_u = p_u - i_u) \implies (\theta_u(i) \in \{v_{i_u}, \cdots, v_{p_u}\})$ est triviale car toutes les variables $v_{i_u}, \cdots, v_{p_u}$ doivent être chargées pour le calcul de u. Par conséquent, il est seulement nécessaire de montrer qu'il existe un ordre d'empilement optimal tel que $K_u = p_u - i_u$. Soit alors θ un ordre d'empilement compatible tel qu'il existe $u \in V$ vérifiant $K_u > p_u - i_u$. Nous allons vérifier que dans les deux cas possibles on peut construire un ordre compatible vérifiant le lemme et conduisant à un coût moindre :

(i) Il existe un prédécesseur v_i de $u, i \leq i_u$ et un entier $k \in \{1, \cdots, K_u\}$ tel que $\theta_u(k) = v_i$. Considérons l'ordre d'empilement θ'_u obtenu en supprimant l'opération $\mathrm{LD}(v_i)$ causée par θ_u.

$$\theta'_u = (\theta_u(1), \cdots, \theta_u(k-1), \theta_u(k+1), \cdots, \theta_u(K_u))$$

On note θ' l'ordre total d'empilement déduit de θ en remplaçant l'ordre d'empilement de v_i :

$$\theta' = \{(v, \theta_v), v \in V - \{u\}\} \cup \{(u, \theta'_u)\}$$

Clairement, θ' est compatible avec φ. Il nous reste donc à prouver que $\mathrm{UCS}_{\theta'}(\varphi) - \mathrm{UCS}_\theta(\varphi) \leq 0$. Soit W, l'ensemble des sommets qui sont compris entre v_i et le sommet

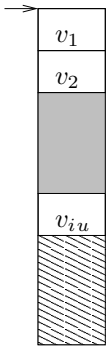


FIG. 2.7 – Exemple pour le lemme 1

de la pile avant l'éventuelle opération $\mathrm{LD}(v_i)$. Le coût de cette opération est $|W|$. Sa suppression a pour effet de diminuer de 1 le futur chargement éventuel d'un variable de W. D'autre part, les coûts de chargement des variables autres que v_i ne sont pas modifiés, c'est-à-dire :

$$\forall w \neq v_i, \mathrm{UCS}_{\theta'}(\varphi, w) \leq \mathrm{UCS}_{\theta}(\varphi, w)$$

Si v_i n'est plus rechargée dans le futur, on a donc clairement $\mathrm{UCS}_{\theta'}(\varphi) \leq \mathrm{UCS}_{\theta}(\varphi)$. Sinon, notons q (resp. q') le coût d'un tel chargement en supposant que θ (resp. θ') a été utilisé. On a $q' \geq q$, et

$$\mathrm{UCS}_{\theta'}(\varphi) - \mathrm{UCS}_{\theta}(\varphi) \leq q' - (q + |W|)$$

La figure 2.8 illustre les états de la pile dans les deux cas juste avant le nouveau chargement de v_i. W' est l'ensemble des éléments de W qui n'ont pas été à nouveau chargés depuis la suppression de l'opération $\mathrm{LD}(v_i)$. Clairement, $q' = q + |W'| \leq q + |W|$, ce qui nous permet de conclure que dans ce cas aussi on a $\mathrm{UCS}_{\theta'}(\varphi) \leq \mathrm{UCS}_{\theta}(\varphi)$.

(ii) Il existe un prédécesseur v_i de $u, i > i_u$ et deux entiers différents k_1 et k_2 dans $\{1, \cdots, K_u\}$ tels que $\theta_u(k_1) = \theta_u(k_2) = v_i$ (c'est-à-dire que v_i est chargé plus d'une fois par θ). Alors il est simple de constater que la suppression du premier chargement ne modifie pas l'état final de la pile.

□

Lemme 2. *L'ordre d'empilement tel que $\theta_u = (v_{i_u+1}, \cdots, v_{p_u})$ pour tout $u \in V$ est optimal.*

Preuve On suppose qu'il existe un ordre d'empilement optimal θ satisfaisant le lemme 1, mais qui ne satisfait pas les conditions du lemme actuel. Montrons qu'on peut construire à partir de celui-ci un ordre compatible avec φ conduisant à un coup moindre, et qu'en répétant l'opération on atteint l'ordre de l'hypothèse. Pour cela, considérons le plus petit entier $k \in \{1, \cdots, p_u - 1\}$ tel que $\theta_u(k) = v_j$, $\theta_u(k+1) = v_i$ et $j > i$ (c'est-à-dire la première inversion par rapport à l'ordre du lemme). On note θ'_u l'ordre obtenu

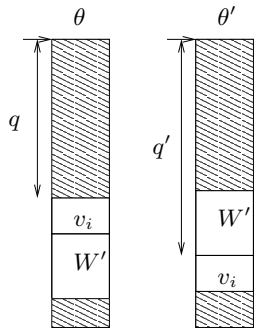


Fig. 2.8 – Pile quand v_i est rechargé (cas de θ et θ')

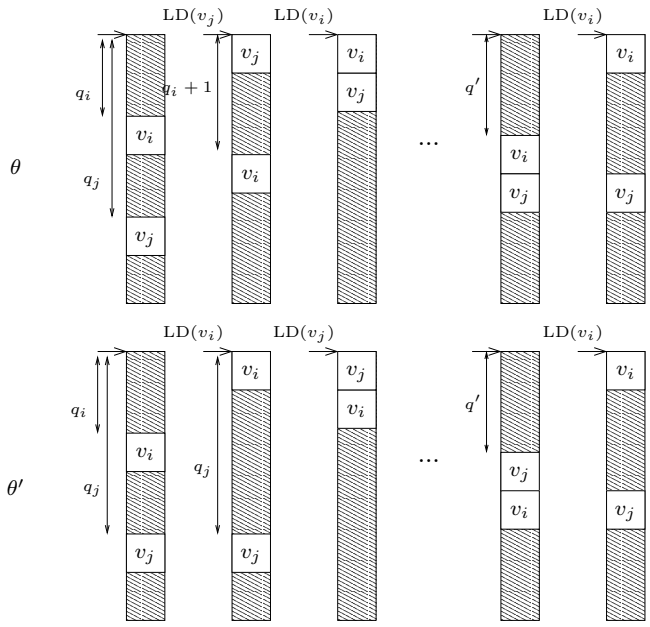


Fig. 2.9 – Comparaison entre les ordres d'empilement θ et θ'

par l'inversion de $\theta_u(k)$ et $\theta_u(k+1)$, et on note θ' l'ordre d'empilement total déduit de θ comme précédemment par $\theta' = \{(v, \theta_v), v \in V - \{u\}\} \cup \{(u, \theta'_u)\}$. Nous allons prouver que $\mathrm{UCS}_{\theta'}(\varphi) \leq \mathrm{UCS}_{\theta}(\varphi)$.

Comparons, comme illustré dans la figure 2.9, les états successifs de la pile lors de deux exécutions selon θ et θ'. Avant que le programme atteigne la paire d'opérations "$\mathrm{LD}(v_j); \mathrm{LD}(v_i)$" de θ (qui correspond à "$\mathrm{LD}(v_i); \mathrm{LD}(v_j)$" dans le cas où θ' est utilisé), les deux suites d'états possibles de la pile sont identiques à chaque pas. Ensuite, les places de v_j et de v_i sont simplement permutées selon l'ordre utilisé, jusqu'à ce qu'une opération $\mathrm{LD}(v_i)$ ou $\mathrm{LD}(v_j)$ soit rencontrée. Lors du reste de l'exécution, les états de la pile ne dépendent pas de l'ordre d'empilement choisi. Par conséquent, la différence de coût entre $\mathrm{UCS}_{\theta'}(\varphi)$ et $\mathrm{UCS}_{\theta}(\varphi)$ est seulement imputable aux trois opérations LD qui viennent d'être énumérées. Le pire cas arrive quand le troisième chargement est $\mathrm{LD}(v_i)$. Soit q' la profondeur de v_j dans la pile pour θ juste avant celui-ci. q' est aussi la profondeur de v_j dans la pile pour θ' au même instant. Par conséquent, on a :

$$\mathrm{UCS}_{\theta}(\varphi) - \mathrm{UCS}_{\theta'}(\varphi) \leq (q_j + q_i + 1) - (q_i + q_j) + q' - (q' + 1) = 0$$

□

Dans la suite, on supposera que pour tout ordre φ, les variables sont toujours chargées selon l'ordre d'empilement total optimal $\theta^\star$ déduit du lemme 2. Par conséquent, le coût d'une numérotation de graphe pourra être notée sans ambiguïté $\mathrm{UCS}(\varphi) = \mathrm{UCS}(\varphi, G) = \mathrm{UCS}_{\theta^\star}(\varphi, G)$.

2.2.5 Formalisation des problèmes

Les deux fonctions de coût présentées ci-dessus donnent lieu à deux problèmes d'optimisation qui sont étudiés dans la suite de la thèse, et dont les versions décisionelles sont les suivantes :

Problème: Minimum uniform cost stack (minUCS)

Instance: $G = (V, A)$ un graphe orienté sans circuit, un entier K.

Question: Existe-t-il une numérotation $\varphi : V \to \{1, ..., |V|\}$ telle que $\mathrm{UCS}(\varphi, G) \leq K$?

Problème: Minimum Directed Sumcut (minDSC)

Instance: $G = (V, A)$ un graphe orienté sans circuit, un entier K.

Question: Existe-t-il une numérotation $\varphi : V \to \{1, ..., |V|\}$ telle que $\mathrm{DSC}(\varphi, G) \leq K$?

2.2.6 Remarques

Pour une numérotation φ donnée, le score $\mathrm{DSC}(\varphi, G)$ peut être déterminé de façon trivialle en $\mathcal{O}(m)$ opérations, et le coût $\mathrm{UCS}(\varphi, G)$ en $\mathcal{O}(mn)$ opérations. En effet, dans ce dernier cas on doit garder la trace de chaque mouvement en mémoire, c'est-à-dire représenter la pile modélisant celle-ci. Cependant, l'utilisation d'arbres AVL [AVL62] permet d'améliorer ce nombre à $\mathcal{O}(m \log n)$ [Rob03]. Ceci nous permet aussi de dire que les deux problèmes de décision correspondants sont dans NP.

Pour les deux fonctions de coût proposées, le problème peut être décomposé de façon naturelle si le graphe de précédence possède plusieurs composantes connexes. Formellement, pour $\mathcal{C} \equiv \mathrm{DSC}$ ou $\mathcal{C} \equiv \mathrm{UCS}$, si le nombre de composantes connexes est k, alors $\min_\varphi \mathcal{C}(\varphi, G) = \sum_{i=1}^{k} \min_{\varphi_i} \mathcal{C}(\varphi_i, G_i)$, où les composantes connexes sont notées $G_1, \cdots, G_k$. Sans perte de généralité, on pourra donc supposer que tous les graphes étudiés sont connexes.

2.2.7 Corrélation entre DSC et UCS

Nous avons montré en 3.2.2 et en 3.2.1 que les problèmes minDSC et minUCS peuvent être résolus en temps polynomial dans le cas des arborescences et des anti-arborescences, et surtout que les algorithmes sont identiques. Or, dans le cas général ce résultat n'est pas établi.

Expérimentalement, nous avons pu observer une corrélation linéaire des deux critères (voir les figures 2.10 et 2.11). Les expériences ont été menées de la façon suivante :

- Génération pseudo-aléatoire de graphes orientés sans circuits, ou éventuellement génération de graphes à partir de description de circuits existants,
- Pour un graphe donné, génération d'un grand nombre de numérotations en utilisant toutes les heuristiques, de façon à avoir une large répartition des scores obtenus,
- Pour chaque numérotation, calcul des scores DSC et UCS.

Un exemple de cette corrélation linéaire peut être observé en figure 2.10. Pour cet exemple, nous avons numéroté le graphe de précédence d'un multiplieur IEEE 64 bits à virgule flottante conçu à l'aide de GenOptim [Hou97]. La figure 2.11 montre un résultat similaire obtenu pour un graphe de 10^6 sommets généré de façon pseudo-aléatoire.

Cette corrélation nous a permis de développer des heuristiques communes pour les deux modèles (chapitre 4), et dans un second temps de les tester simultanément.

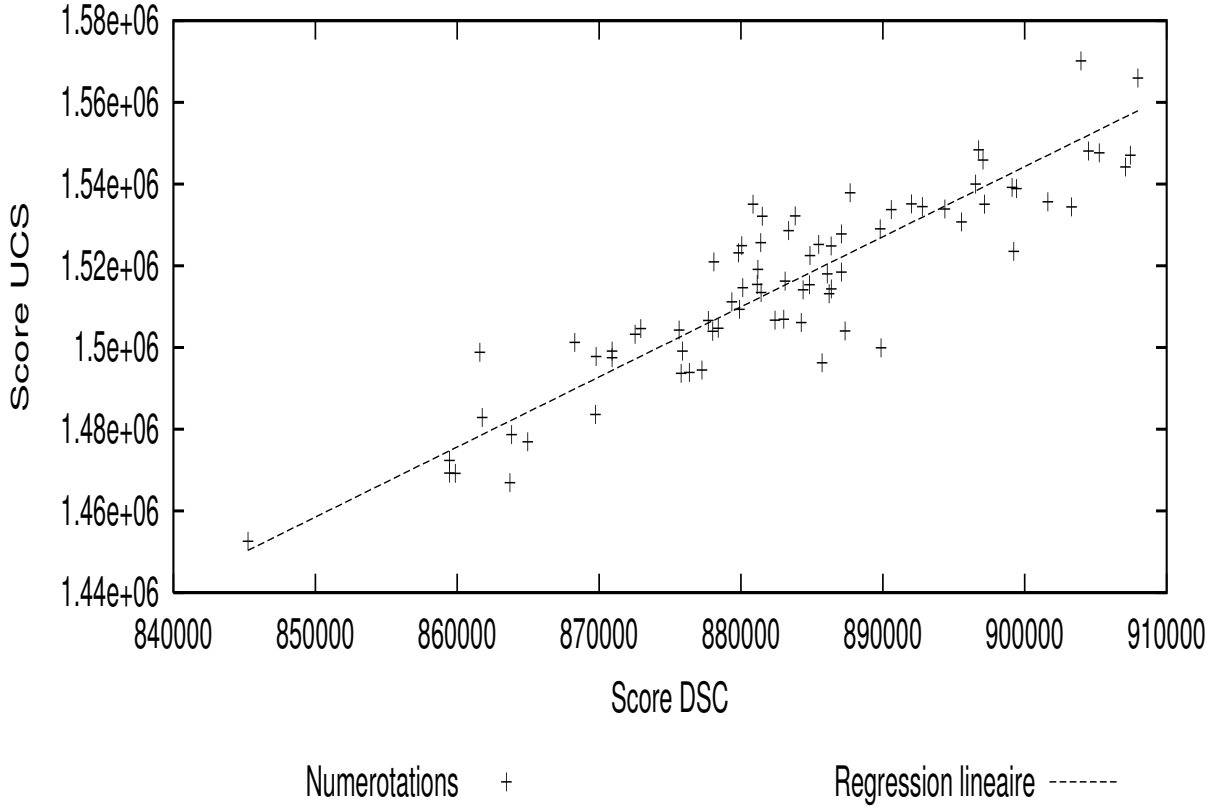


Fig. 2.10 – Corrélation expérimentale entre les scores DSC et UCS (multiplieur IEEE)

2.3 Conclusion

Dans ce chapitre, après avoir montré de façon succincte le fonctionnement de la mémoire lors de l'exécution d'un programme sur un ordinateur, nous avons présenté une famille de problèmes d'optimisation combinatoire : les problèmes de numérotation de graphes. C'est dans ce cadre que nous avons défini et formalisé deux modèles susceptibles de représenter simplement les états d'une mémoire lors de l'exécution d'un programme. Il est important de répéter que c'est dans la simplicité des modèles que réside leur intérêt. En effet, on rappelle que l'objectif est de réarranger un code de simulation afin de gagner du temps sur l'exécution du programme compilé. Un compromis est réalisé car la durée de la phase de réarrangement n'est pas négligeable, mais elle ne doit pas devenir un frein, ce qui risquerait d'arriver avec des modèles de mémoire plus réalistes. De plus, comme il est montré dans le chapitre 4, il s'avère que le réalisme de nos modèles est suffisant pour obtenir des résultats.

Dans le chapitre suivant, nous commençons par montrer que les deux sont difficiles

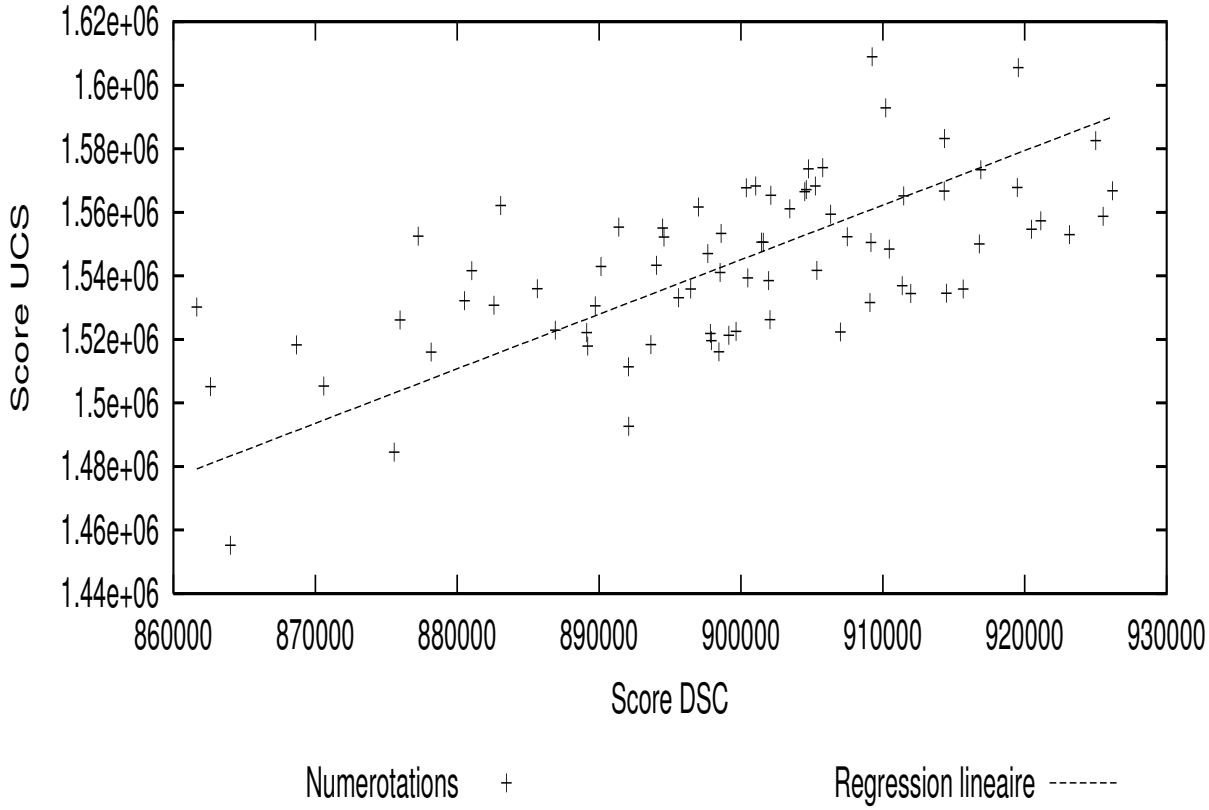


Fig. 2.11 – Corrélation expérimentale entre les scores DSC et UCS (graphe aléatoire de 10^6 sommets)

pour les graphes de profondeur 2, puis nous montrons quelques classes de graphes pour lesquelles des algorithmes simples existent.

Chapitre 3

Résultats théoriques

Dans le chapitre 1, nous avons montré comment un programme de simulation au cycle près peut être représenté par le graphe de causalité de ses variables. Puis, dans le chapitre 2, nous avons présenté deux modèles destinés à représenter, pour un graphe orienté sans circuit donné, une estimation relative du temps d'exécution du programme correspondant compilé. Dans ce chapitre sont présentés les résultats théoriques correspondants.

La section 3.1 est dédiée à la présentation des résultats de complexité. Nous montrons que la détermination d'une numérotation optimale pour les scores DSC et UCS est un problème difficile pour les graphes de profondeur 2.

Puis, en section 3.2, nous présentons des algorithmes permettant d'atteindre des numérotations optimales pour quatre classes de graphes en un nombre polynomial d'opérations. Tout d'abord, dans les sections 3.2.1 et 3.2.2, nous montrons que les problèmes minDSC et minUCS peuvent être résolus en temps polynomial pour les arborescences et les anti-arborescences. Ensuite, en section 3.2.3, nous utilisons la construction récursive de certains graphes série-parallèles pour concevoir un algorithme polynomial permettant de minimiser le score DSC dans ce cas. Nous montrons aussi grâce à des contre-exemples que notre méthode ne peut s'appliquer à deux autres classes plus grandes de graphes série-parallèles.

Le dernier résultat, présenté en section 3.2.4, montre que, grâce à une relation d'ordre total sur l'espace quotient des sommets sans successeurs (appelés sommets puits) inspirée de la caractéristation des ordres intervalles présentée en [PY79], il est possible de résoudre le problème minDSC en temps polynomial pour les ordres intervalles.

3.1 Graphes de profondeur 2

Cette section présente les résultats de complexité correspondant aux deux fonctions de coût introduites au chapitre 2. Après avoir formalisé les problèmes pour les graphes de profondeur deux, nous montrons qu'une même réduction polynomiale permet de prouver la NP-complétude des deux.

3.1.1 Formalisation des problèmes

Dans cette section, nous montrons que les problèmes minBipDSC et minBipUCS (formalisés ci-dessous) sont NP-complets. Dans les deux cas, notre preuve repose sur une réduction polynomiale du problème minLA (Minimum Linear Arrangement) présenté précédemment et rappelé ci-dessous. Nous conluons ensuite grâce à l'appartenance à NP de ces deux problèmes, montrée en 2.2.6.

Problème: Minimum Bipartite uniform cost stack (minBipUCS)

Instance: $G = (V, A)$ un graphe orienté biparti sans circuit, un entier K.

Question: Est-il possible de trouver une numérotation $\varphi : V \to \{1, ..., |V|\}$ telle que $\mathrm{UCS}(\varphi, G) \leq K$?

Problème: Minimum Bipartite Directed Sumcut (minBipDSC)

Instance: $G = (V, A)$ un graphe orienté biparti sans circuit, un entier K.

Question: Est-il possible de trouver une numérotation $\varphi : V \to \{1, ..., |V|\}$ telle que $\mathrm{DSC}(\varphi, G) \leq K$?

Problème: Minimum Linear Arrangement (minLA) [GJ79]

Instance: $G = (V, X)$ un graphe non orienté, et un entier positif K.

Question: Existe-t-il une numérotation f telle que $\sum_{\{u,v\} \in X} |f(u) - f(v)| \leq K$?

Le théorème 3.1.1 montre que minLA est difficile :

Théorème 3.1.1. *minLA est NP-complet [GJ79].*

Soit I une instance de minLA donnée par un graphe $G = (V, X)$ et un entier K, $n = |V|$, $m = |X|$. Dans les preuves de complexité des problèmes minDSC et minUCS, la réduction repose sur une transformation identique. On considère une instance I' de minDSC et minUCS définie par un graphe $G' = (V'; X')$ biparti décrit de la manière suivante :

- On note A et B les parties de V'. c'est-à-dire que pour tout arc $(u, v) \in X'$, on a $u \in A$ et $v \in B$.
- $B = V$.
- A chaque arête $e = \{u, v\} \in X$ on associe :
  - Un sommet $x_e \in A$,
  - 2 arcs (x_e, u) et (x_e, v).
- A chaque sommet $u \in B$ on ajoute un ensemble $E_u \subset A$ de n^6 prédécesseurs.
- La valeur du coût est inférieure à une borne qui dépend du problème (voir par la suite).

Dans la figure 3.1, on a pour I d'une part $n = 4$, $m = 5$. Pour I', on a $|A| = m + n^7$, $|B| = m$.

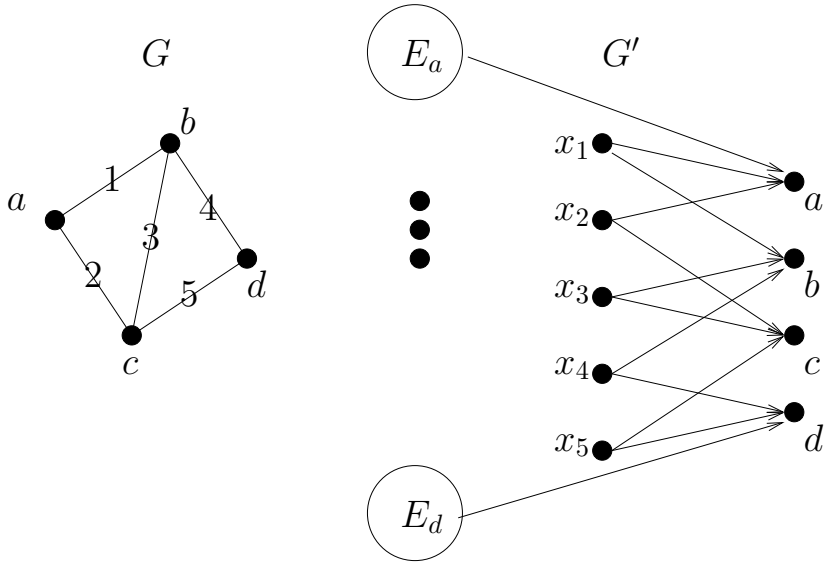


Fig. 3.1 – Construction du graphe G' associé au graphe G définissant une instance de minLA

Cette transformation est clairement polynomiale et le graphe G' est biparti.

3.1.2 NP-complétude de minBipDSC

La preuve de complexité du problème minBipDSC repose sur une propriété préliminaire. Soit $G = (A, B; E)$ un graphe biparti. Soit φ une numérotation des sommets de $A \cup B$, et soit f une numérotation des sommets de B.

Propriété 1. *A partir d'une numérotation f des sommets de B, on peut construire une numérotation φ de G en numérotant chaque sommet de A juste avant son premier successeur. Pour f fixée, φ ainsi construite minimise alors le critère DSC.*

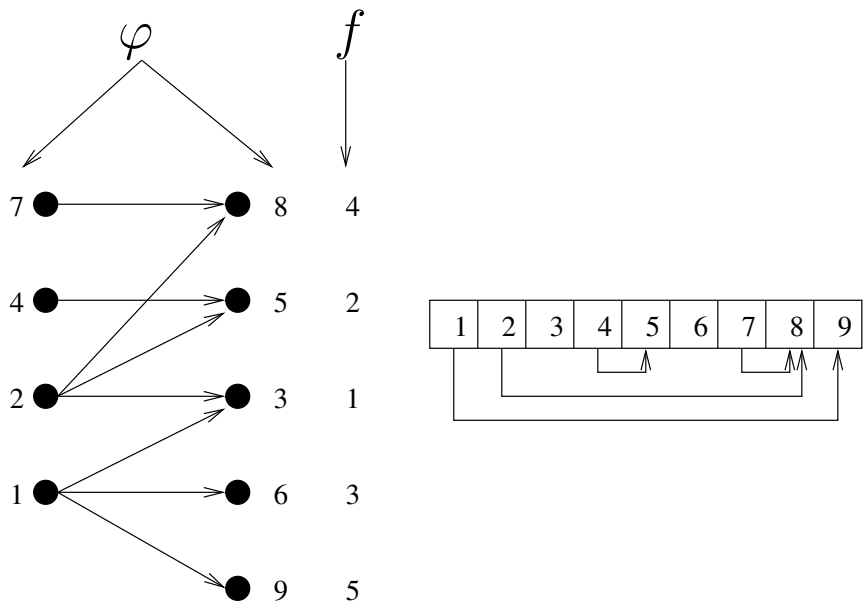


FIG. 3.2 – Exemple de numérotations des sommets d'un graphe biparti

Preuve Soit φ une numérotation qui ne vérifie pas la propriété. Alors, il existe un sommet $i \in A$ qui a un ou plusieurs sommets de $A \cup B$ entre lui et son premier successeur. Soient $j_1 \cdots j_k$ les successeurs de i numérotés dans l'ordre correspondant à φ, et soit C l'ensemble des sommets numérotés entre i et j_1. On construit φ' en échangeant i et C. On veut montrer que $\mathrm{DSC}(G, \varphi') \leq \mathrm{DSC}(G, \varphi)$.

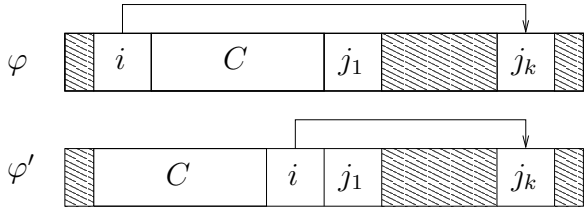


FIG. 3.3 – Transformation de φ à φ'

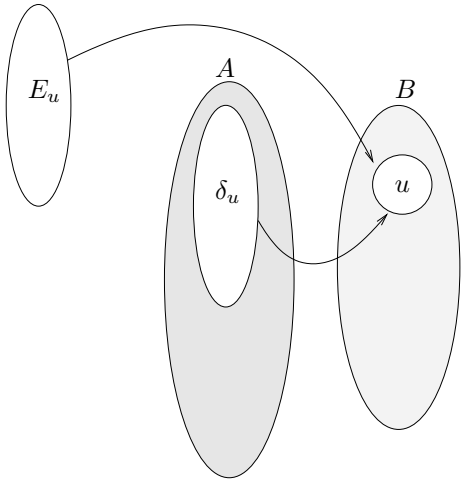


Fig. 3.4 – Exemple pour la notation δ_u

(i) $\mathrm{DSC}(i,\varphi) - \mathrm{DSC}(i,\varphi') = |C|$.
(ii) Soit un sommet $j \in A \cup B$ tel que $\mathrm{DSC}(j,\varphi') > \mathrm{DSC}(j,\varphi)$.
  – Si $j \in C$, alors son dernier successeur n'est pas dans C. On a alors $\mathrm{DSC}(j,\varphi') - \mathrm{DSC}(j,\varphi) = 1$ car j a été déplacé d'une unité.
  – Sinon, $j \notin C$, donc son coût n'a pas été augmenté.

Il y a au plus $|C|$ éléments qui vérifient le deuxième cas, donc

$$\sum_{j\in C}(\mathrm{DSC}(j,\varphi') - \mathrm{DSC}(j,\varphi)) \leq |C|.$$

Ceci implique

$$\begin{aligned}\mathrm{DSC}(G,\varphi') - \mathrm{DSC}(G,\varphi) &= \mathrm{DSC}(i,\varphi') - \mathrm{DSC}(i,\varphi)\\ &+ \sum_{j\in C}(\mathrm{DSC}(j,\varphi') - \mathrm{DSC}(j,\varphi))\\ &+ \sum_{j\in\{A\cup B\}-C-\{i\}}(\mathrm{DSC}(j,\varphi') - \mathrm{DSC}(j,\varphi))\\ &\leq -|C| + |C| = 0.\end{aligned}$$

Enfin, on peut remarquer que si i et j sont 2 sommets de A avec $\varphi(j) = \varphi(i) + 1$, alors si on les échange cela ne modifie pas la fonction de coût. Par conséquent, quand on a le choix pour numéroter plusieurs sommets de A, on peut les choisir au hasard. □

Pour tout $u \in B$, on note δ_u l'ensemble des prédécesseurs de u (pour une certaine numérotation φ) effectués juste avant u qui ne sont pas dans E_u (figure 3.4). Il est aisé de constater que pour tout $u \in B$, on a $|\delta_u| \leq m$. A partir de cette borne on peut déduire le lemme technique suivant :

Lemme 3.

$$\sum_{u\in V}\sum_{\alpha=1}^{|\delta_u|}\alpha \leq \frac{1}{2}n(m^2+m)$$

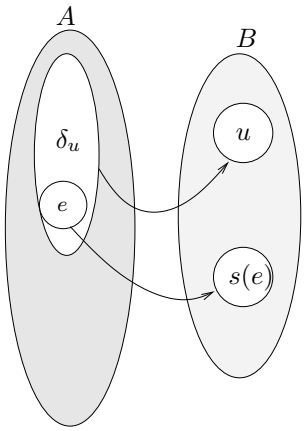


FIG. 3.5 – Exemple pour la notation $s(e)$

Preuve

$$\begin{aligned} \sum_{u\in V}\sum_{\alpha=1}^{|\delta_u|}\alpha &= \frac{1}{2}\sum_{u\in V}(|\delta_u|(|\delta_u|+1)) \\ &= \frac{1}{2}\sum_{u\in V}(|\delta_u|^2+|\delta_u|) \\ \text{Or, } \forall u\in V, |\delta_u|\leq m, \text{ donc } \sum_{u\in V}\sum_{\alpha=1}^{|\delta_u|}\alpha &\leq \frac{1}{2}\sum_{u\in V}(m^2+m) \\ &\leq \frac{1}{2}n(m^2+m). \end{aligned}$$

Théorème 3.1.2. *Il existe une transformation polynomiale de minLA à minBipDSC.*

Preuve On considère l'instance I de minLA et l'instance I' de minDSC décrites en 3.1.1. La valeur de la borne pour cette transformation est égale à (K restant à définir) :

$$D = n^6 m + \frac{n^7(n^6+1)}{2} + m^2 + n^6 K + \frac{1}{2}n(m^2+m)$$

Soit $u \in V$. Pour tout $e \in \delta_u$, on note $s(e)$ le successeur de e dans G' tel que $\mathrm{DSC}(e,\varphi) = \varphi(s(e)) - \varphi(e)$ (figure 3.5).

(i) Supposons que la réponse au problème de minLA est "oui", et soit f une solution de minLA. On peut construire une numérotation φ de G' à partir de f de la manière suivante :

(a) Les sommets de B tels que $f(1) < f(2) < \cdots < f(n)$ sont numérotés de façon à ce que $\varphi(1) < \varphi(2) < \cdots < \varphi(n)$.

(b) Les sommets de A sont numérotés juste avant leur premier successeur. De plus, pour tout sommet $u \in B$, on numérote les sommets de E_u avant ceux de δ_u. On obtient alors

$$\begin{aligned} \mathrm{DSC}(G', \varphi) &= \sum_{v \in A} \mathrm{DSC}(v, \varphi) \\ &= \sum_{u \in V} \mathrm{DSC}(E_u, \varphi) + \sum_{u \in V} \sum_{e \in \delta_u} \mathrm{DSC}(e, \varphi) \end{aligned}$$

– Pour tout $u \in V$,

$$\begin{aligned} \mathrm{DSC}(E_u, \varphi) &= \sum_{\alpha=1}^{n^6} (\alpha + |\delta_u|) \\ &= n^6 |\delta_u| + \frac{n^6(n^6+1)}{2}. \\ \text{Donc, } \sum_{u \in V} \mathrm{DSC}(E_u, \varphi) &= n^6 \sum_{u \in V} |\delta_u| + n \frac{n^6(n^6+1)}{2} \\ &= n^6 m + \frac{n^7(n^6+1)}{2}. \end{aligned}$$

– Pour tout $e \in \delta_u$, $\exists \alpha \in \{1, \cdots, |\delta_u|\}$ avec $\varphi(u) = \varphi(e) + \alpha$. De plus, $\mathrm{DSC}(e, \varphi) = \varphi(s(e)) - \varphi(e)$ (on rappelle que $s(e)$ désigne le dernier successeur de e). Donc :

$$\begin{aligned} \sum_{e \in \delta_u} \mathrm{DSC}(e, \varphi) &= \sum_{e \in \delta_u} (\varphi(s(e)) - \varphi(e)) \\ &= \sum_{e \in \delta_u} (\varphi(s(e)) - \varphi(u)) + \sum_{\alpha=1}^{|\delta_u|} \alpha \end{aligned}$$

Or, par construction de φ, $|\varphi(s(e)) - \varphi(u)| \leq m + n^6 |f(s(e)) - f(u)|$, donc $\sum_{e \in \delta_u} (\varphi(s(e)) - \varphi(u)) \leq m^2 + n^6 \sum_{e \in \delta_u} |f(s(e)) - f(u)|$. Or, pour tout $e = \{u, v\} \in X$, avec $e \in \delta_u$, on a $s(e) = v$, donc :

$$\sum_{u \in V} \sum_{e \in \delta_u} |f(s(e)) - f(u)| = \sum_{e = \{u,v\} \in X} |f(u) - f(v)| = K$$

donc $\sum_{e \in \delta_u} (\varphi(s(e)) - \varphi(u)) \leq m^2 + n^6 K$.
En appliquant le lemme 3, on obtient

$$\mathrm{DSC}(G', \varphi) \leq n^6 m + \frac{n^7(n^6+1)}{2} + m^2 + n^6 K + \frac{1}{2} n(m^2 + m) = D$$

(ii) Réciproquement, supposons qu'on a une solution φ de l'instance I' de minBipDSC. Ceci signifie que $\mathrm{DSC}(\varphi, G) \leq D$. On construit une numérotation f des sommets de G telle que $\forall a, b \in V^2, f(a) < f(b) \Leftrightarrow \varphi(a) < \varphi(b)$. On peut supposer sans perte de

généralité que φ vérifie la propriété 1, et que pour tout $u \in V$, les éléments de E_u sont numérotés avant ceux de δ_u.
Comme précédemment,

$$\begin{aligned}\mathrm{DSC}(G', \varphi') &= \sum_{u \in V} \mathrm{DSC}(E_u, \varphi) + \sum_{u \in V} \sum_{e \in \delta_u} \mathrm{DSC}(e, \varphi) \\ &= n^6 m + \frac{n^7(n^6+1)}{2} + \sum_{u \in V} \sum_{e \in \delta_u} (\varphi(s(e)) - \varphi(u)) \\ &\quad + \sum_{u \in V} \sum_{\alpha=1}^{|\delta_u|} \alpha\end{aligned}$$

Or, $\varphi(s(e)) - \varphi(u) = |\varphi(s(e)) - \varphi(u)| \geq n^6 |f(s(e)) - f(u)|$, donc $\sum_{u \in V} \sum_{e \in \delta_u} \varphi(s(e)) - \varphi(u) \geq n^6 \sum_{u \in V} \sum_{e \in \delta_u} |f(s(e)) - f(u)|$. Comme précédemment, pour tout $e = \{u, v\} \in X$, avec $e \in \delta_u$, $s(e) = v$. Donc

$$\sum_{u \in V} \sum_{e \in \delta_u} |f(s(e)) - f(u)| = \sum_{e=\{u,v\} \in X} |f(v) - f(u)|.$$

Et donc :

$$\sum_{u \in V} \sum_{e \in \delta_u} (\varphi(s(e)) - \varphi(u)) \geq n^6 \sum_{e=\{u,v\} \in X} |f(v) - f(u)|.$$

Ainsi, comme $\mathrm{DSC}(G', \varphi') \leq D$,

$$n^6 m + \frac{n^7(n^6+1)}{2} + \sum_{u \in V} \sum_{e \in \delta_u} (\varphi(s(e)) - \varphi(u)) + \sum_{u \in V} \sum_{\alpha=1}^{|\delta_u|} \alpha \leq D$$

Puis, en reportant la définition de D, on obtient que

$$n^6 \sum_{e=\{u,v\} \in X} |f(v) - f(u)| + \sum_{u \in V} \sum_{\alpha=1}^{|\delta_u|} \alpha \leq m^2 + n^6 K + \frac{1}{2} n(m^2 + m)$$

Et comme $\sum_{u \in V} \sum_{\alpha=1}^{|\delta_u|} \alpha \geq 0$:

$$\begin{aligned} n^6 \sum_{e=\{u,v\} \in X} |f(v) - f(u)| &\leq m^2 + \frac{1}{2} n(m^2 + m) + n^6 K \\ &\leq n^4 + \frac{1}{2} n(n^4 + n^2) + n^6 K \\ \sum_{e=\{u,v\} \in X} |f(v) - f(u)| &\leq K + \frac{1}{n^2} + \frac{1}{2n} + \frac{1}{2n^3} \\ \Longrightarrow \sum_{e=\{u,v\} \in X} |f(v) - f(u)| - K &\leq \frac{1}{n^2} + \frac{1}{2n} + \frac{1}{2n^3} < 1, \forall n > 1 \end{aligned}$$

D'où le résultat.

□

Corollaire 1. *La version d'optimisation de minBipDSC est NP-difficile pour les graphes orientés sans circuit bipartis.*

3.1.3 NP-complétude de minBipUCS

Dans cette section, nous montrons que le problème minBipUCS est NP-complet, à partir du problème Minimum Linear Arrangement.

Théorème 3.1.3. *Il existe une transformation polynomiale de minLA à minBipUCS.*

Preuve

On considère l'instance I de minLA et l'instance I' de minDSC décrites en 3.1.1. La valeur du coût pour cette transformation est inférieure à :

$$D = (n^6 + m + 1)K + m^2$$

(i) Supposons que la réponse au problème de minLA est "oui", et soit f une solution. Dans le but de simplifier les notations, on peut supposer sans perte de généralité que $f(v) = i$ pour tout v de V. Dans le même but, on note y_i le sommet de B tel que $f(v) = i$. Une numérotation φ de l'instance associée du problème de minUCS est construite en numérotant les sommets y_i de B selon la numérotation f de X. Plus simplement, $\varphi(y_1) < \varphi(y_2) < ... < \varphi(y_n)$. En supposant que $\varphi(y_0) = 0$ (où y_0 est un sommet fictif), on peut alors numéroter les éléments de A comme suit :
- de $\varphi(y_{i-1}) + 1$ à $\varphi(y_{i-1}) + n^6$, les n^6 éléments de E_{y_i},
- de $\varphi(y_{i-1}) + n^6 + 1$ à $\varphi(y_i) - 1$, les éléments $x(\{i, j\})$ de A n'appartenant à aucun E et tels que $i < j$. Clairement, les sommets $x(\{i, j\})$ avec $j < i$ ont été numérotés avant $\varphi(y_j)$.

Nous voulons maintenant prouver que $\mathrm{UCS}(\varphi, G) \leq K$. On rappelle que pour toute arc $(x, y) \in X'$, $\mathrm{UCS}(\varphi, (x, y))$ représente le coût de chargement de x pour l'évaluation de y. L'expression peut alors être transformée en :

$$\mathrm{UCS}(\varphi, G) = \sum_{y \in B} \sum_{x \in \Gamma^-(y)} \mathrm{UCS}(\varphi, (x, y))$$

Dans cette somme, les coûts $\mathrm{UCS}(\varphi, (x, y))$ ne sont non nuls que pour les valeurs $y = y_i$ et $x = x(\{i, j\})$ tels que $j < i$. On considère donc un sommet x vérifiant cette propriété. Comme x est un prédécesseur de y_i, il a été précédemment placé dans la pile lors d'une opération RD pour le calcul de y_j. En effet, $j < i$ implique par construction que $\varphi(y_j) < \varphi(y_i)$. On peut donc écrire $\mathrm{UCS}(\varphi, (x, y_i))$ sous la forme

$$\mathrm{UCS}(\varphi, (x, y_i)) = \Delta(x, y_j) + \Delta(y_j),$$

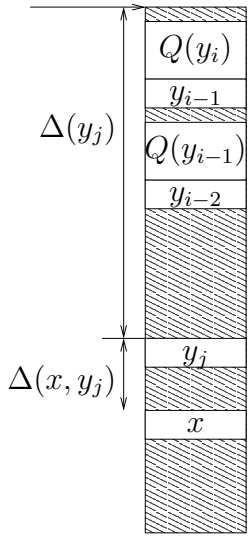


FIG. 3.6 – Calcul de y_i

où $\Delta(x, y_j)$ (*resp.* $\Delta(y_j)$) est le nombre de sommets empilés entre x et y_j, y_j inclus (*resp.* entre y_j et le sommet de la pile) juste avant le calcul de y_i (voir figure 3.6). Chaque sommet $y_j \in Y$ a au plus m prédécesseurs x dans A et n'appartenant à aucun E, donc $\Delta(x, y_j) \leq m$. De plus, tout sommet $y_k \in B$ a au plus $n^6 + m$ prédécesseurs. Comme les sommets de Y sont empilées selon la numérotation φ, on a donc $\Delta(y_j) \leq (f(i) - f(j))(n^6 + m + 1)$.

$$\mathrm{UCS}(\varphi, G) \leq m^2 + (n^6 + m + 1) \sum_{\{i,j\} \in X} |f(i) - f(j)| \leq m^2 + (n^6 + m + 1) \times K$$

On obtient par conséquent que $\mathrm{UCS}(\varphi, G) \leq (n^6 + m + 1)K + m^2 = D$. φ est donc une solution pour l'instance I' de minBipUCS.

(ii) Réciproquement, supposons qu'on a une solution φ de l'instance I' de minBipUCS. Ceci signifie que $\mathrm{UCS}(\varphi, G) \leq D$. On peut supposer sans perte de généralité que les sommets de X sont numérotés de façon à ce que $\varphi(y_1) < \varphi(y_2) < \cdots < \varphi(y_n)$. On construit alors la numérotation f de V, et nous nous proposons de démontrer que la fonction $f \equiv \mathrm{Id}$ est une solution de minLA pour l'instance I.
Considérons l'état de la pile juste avant une opération LD pour le calcul d'un sommet y_j. Comme aucun des sommets y_k, $k \in \{1, \cdots, j-1\}$ n'ont été réutilisés depuis leur création, les éléments de A apparaissent dans la pile dans l'ordre de leurs indices. Tout d'abord, on remarque que pour chaque valeur $k \in \{2, \cdots, n\}$, les éléments de E_{y_k} sont empilés juste après y_{k-1}. En effet, comme ces sommets n'ont pas été rechargés, ils sont empilés entre y_{k-1} and y_k. De plus, on suppose qu'il existe un sommet $x_l \in A$ n'appartenant à aucun E et tel que

$$\varphi(y_{k-1}) < \varphi(x(l)) < \varphi(z^\star),$$

où $z^\star$ est l'élément de E_{y_k} de valeur $\varphi(z^\star)$ maximale. Puisque $z^\star$ n'est plus rechargé par la suite, l'échange de $\varphi(x_l)$ et $\varphi(z^*)$ diminuera forcément le coût UCS.

Soit maintenant une arête $e = \{i, j\} \in X$ telle que $i < j$. Au moment du calcul de y_j, x_l est rechargée. Or, la profondeur de x_l est plus grande que celle de y_j, qui vaut au moins $(n^6+1)(j-i)$ (d'après la remarque ci-dessus). Par conséquent, si on utilise $f \equiv \mathrm{Id}$, on a :

$$\mathrm{UCS}(\varphi, (x_l, y_j)) \geq (f(j) - f(i))(n^6 + 1)$$

Par conséquent,

$$\begin{aligned} \mathrm{UCS}(\varphi, G) &= \sum_{e=\{i,j\} \in X, i<j} \mathrm{UCS}(\varphi, (x_l, y_j)) \\ &\geq \sum_{\{i,j\} \in X, i<j} (f(j) - f(i))(n^6+1) = \sum_{\{i,j\} \in X} |f(j) - f(i)|(n^6+1) \end{aligned}$$

Puisque $\mathrm{UCS}(\varphi, G') \leq D$ et $D = (n^6 + m + 1) \times K + m^2$, on en déduit que

$$\left(\sum_{\{i,j\} \in X} |f(j) - f(i)| - K \right) \times (n^6 + 1) \leq mK + m^2$$

De plus, $m \leq n^2$ et $K \leq nm \leq n^3$, donc

$$\sum_{\{i,j\} \in X} |f(j) - f(i)| - K \leq \frac{1}{n} + \frac{1}{n^2} < 1, \forall n > 1$$

et donc $\sum_{\{i,j\} \in X} |f(j) - f(i)| \leq K$. f est donc une solution de I.

□

Corollaire 2. *La version d'optimisation de minBipUCS est NP-difficile pour les graphes orientés sans circuit bipartis.*

3.2 Cas polynômiaux

Dans cette partie, nous montrons que les problèmes minUCS et minDSC définis au chapitre 2 peuvent être résolus en temps poynômial pour certaines classes de graphes.

3.2.1 Anti-arborescences

Dans cette partie, nous supposons que $G = (V, A)$ est une anti-arborescence. Par conséquent, on considère dans la suite que tout sommet $v \in V$ a au plus un arc sortant. Dans un premier temps, nous montrons que le problème minDSC est alors équivalent à un problème d'ordonnancement polynomial. Ensuite nous prouvons la même chose pour le problème minUCS.

Polynomialité de minDSC pour les anti-arborescences

Le fait que tout sommet d'une anti-arborescence a au plus un arc sortant permet d'obtenir une expression très simple de la fonction de coût DSC :

$$\mathrm{DSC}(\varphi, G) = \sum_{(u,v)\in A} \varphi(v) - \varphi(u)$$

Cette formulation montre que, lorsque le graphe de précédence est une anti-arborescence, le coût DSC est égal au coût de l'arrangement linéaire, un problème dont la résolution polynomiale a été montrée dans [AH73]. On peut aussi remarquer que ce problème est un sous-cas du problème d'ordonnancement $1|intree, p_i = 1|\sum w_i C_i$, dans lequel chaque tâche i correspond à un sommet v dans V, le graphe de précédence est égal à G, et les poids des tâches sont les $w_i = \delta^-(v_i)$. Ce problème est polynomial aussi (voir par exemple [Bru98]).

Polynomialité de minUCS pour les anti-arborescences

Pour tout sommet $u \in V$, on rappelle que $\Gamma^{*-}(u)$ représente l'ensemble des ancêtres de u. On note $s(u)$ l'unique successeur de u dans G. On note aussi $G(u)$ le sous arbre de G enraciné en u, et r la racine de G.

Lemme 4. *Pour toute numérotation φ, on peut construire une autre numérotation φ' telle que $UCS(\varphi', G) \leq UCS(\varphi, G)$, et telle que tous les ancêtres de chaque sommet u de V sont numérotés par φ' juste avant u :*

$$\forall v \in \Gamma^{*-}(u), \varphi'(v) \in \{\varphi'(u) - |\Gamma^{*-}(u)|, \cdots, \varphi'(u) - 1\}$$

Preuve On suppose que φ est une numérotation optimale qui ne vérifie pas les conditions du lemme. Soit u le premier (au sens de φ) sommet de V qui ne vérifie pas la condition. Soit k le dernier (toujours au sens de φ) sommet tel que $\varphi(k) < \varphi(u)$ et $k \notin \Gamma^{*-}(u)$.

Par construction (minimalité de $\varphi(u)$), les prédécesseurs de k sont numérotés juste avant k. Pour des raisons de clarté, on définit les ensembles suivants (voir la figure 3.7) :

$$\begin{aligned}
\Omega_1 &= \{v \in V, \varphi(v) < \varphi(k) - |\Gamma^{*-}(k)|\} \\
\Omega_2 &= \{v \in V, \varphi(k) < \varphi(v) < \varphi(u)\} \\
\Omega_3 &= \{v \in V, \varphi(u) < \varphi(v)\}
\end{aligned}$$

On remarque que $s(u)$ et $s(k)$ appartiennent tous les deux à Ω_3. La numérotation φ' est obtenue en déplaçant k et $\Gamma^{*-}(k)$ juste après u (figure 3.7).

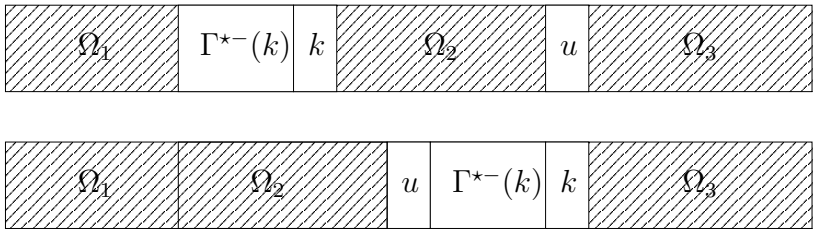


Fig. 3.7 – Transformation de la numérotation pour le coût d'une anti-arborescence

Pour tout $v \in V$, on définit $\Delta(v) = \mathrm{UCS}(\varphi', (v, s(v))) - \mathrm{UCS}(\varphi, (v, s(v)))$. On se propose maintenant de prouver que $\sum_{v \in V-\{r\}} \Delta(v) \leq 0$. La valeur $\Delta(v)$ dépend en fait de 6 cas :

1. Si $v = u$, alors au pire les valeurs de $\{k\} \cup \Gamma^{*-}(k)$ sont encore entre u et le sommet de la pile au moment de l'exécution de $s(u)$ selon φ'. Par conséquent on obtient que $\Delta(u) \leq 1 + |\Gamma^{*-}(k)|$.
2. Si $v = k$, alors k est rapproché de son successeur par φ', donc $\Delta(k) \leq 0$.
3. Si $v \in \Gamma^{*-}(k)$, alors $\Delta(v) = 0$ car de tels sommets sont déplacés avec k.
4. Si $v \in \Omega_2$, alors $\Delta(v) = 0$ car $s(v) \in \Omega_2 \cup \{u\}$.
5. Si $v \in \Omega_3$, alors $\Delta(v) = 0$ car $s(v) \in \Omega_3$.
6. Si $v \in \Omega_1$, alors si $s(v) \in \Omega_1$, $\Delta(v) = 0$. Maintenant, par hypothèse, $\Omega_1 \cap \Gamma^{*-}(u) \neq \emptyset$. Donc il existe $l \in \Omega_1$ avec $s(l) \in \Omega_2 \cup \{u\}$. Pour cette valeur, $\Delta(l) = -1 - |\Gamma^{*-}(k)|$. On en déduit que $\sum_{v \in \Omega_1} \Delta(v) \leq -1 - |\Gamma^{*-}(k)|$.

Ainsi,

$$\sum_{v \in V-\{r\}} \Delta(v) = \Delta(u) + \Delta(k) + \sum_{v \in \Omega_1} \Delta(v) \leq 0$$

□

On considère maintenant une numérotation φ vérifiant les hypothèses du lemme précédent, ainsi qu'un sommet $u \in V$ dont les prédécesseurs sont notés $p_1, ..., p_q$ de façon à ce que $\varphi(p_1) < ... < \varphi(p_q)$. Si on note $|G(u)|$ le nombre de sommets de $G(u)$, le coût de $G(u)$ est :

$$\mathrm{UCS}(\varphi, G(u)) = \sum_{j=1}^{q} \mathrm{UCS}(\varphi, G(p_j)) + \sum_{j=1}^{q} (j-1)|G(p_j)|$$

Cette valeur est minimale si et seulement si $|G(p_1)| \geq |G(p_2)| \geq \cdots \geq |G(p_q)|$. Par conséquent, on en déduit le théorème suivant :

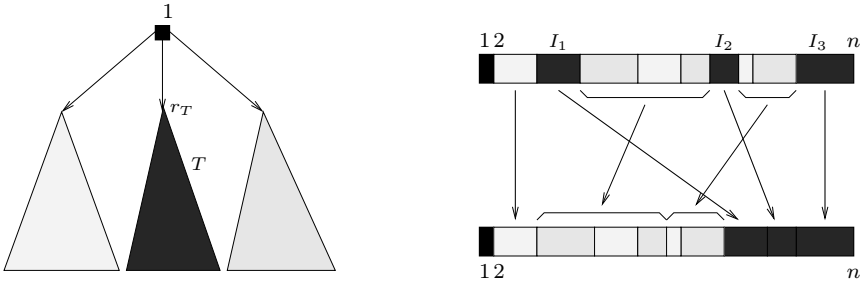


FIG. 3.8 – Re-numérotation d'une arborescence

Théorème 3.2.1. *Toute numérotation φ vérifiant les hypothèses du lemme 4 et telle que pour tout $u \in V$ et pour tout $(p_1, p_2) \in \Gamma^-(u) \times \Gamma^-(u), \varphi(p_1) < \varphi(p_2) \Rightarrow |G(p_1)| \geq |G(p_2)|$ est optimale.*

Comme à chaque étape $u \in V$, les valeurs $|G(v)|, v \in \Gamma^-(u)$ doivent être triées, on obtient un algorithme de complexité $\mathcal{O}(n \log n)$.

3.2.2 Arborescences

Dans un premier temps, nous montrons que dans le cas où G est une arborescence, le problème minDSC peut être résolu en temps polynomial. Ensuite, nous prouvons une relation entre les fonctions DSC et UCS qui nous permet de conclure que l'algorithme permettant de résoudre le problème minDSC dans le cas où G est une arborescence conduit aussi à une numérotation optimale pour la fonction de coût UCS.

Polynomialité de minDSC pour les arborescences

Le graphe $G = (V, A)$ est maintenant une arborescence, ce qui signifie que tout sommet $v \in V$ a au plus un arc entrant. De plus, on a $m = n-1$. L'algorithme que nous proposons permet de construire en temps linéaire une numérotation optimale de G, et repose sur le lemme qui suit.

Lemme 5. *Il existe une numérotation optimale telle que tous les sommets d'un sous-arbre T de la racine de G sont numérotés après les sommets de $G \backslash T$.*

$$\forall u \in T, \forall v \in G \backslash T, \varphi(u) > \varphi(v)$$

Preuve Considérons une numérotation φ qui ne vérifie pas le lemme. Par la transformation décrite en figure 3.8, nous construisons une nouvelle numérotation φ' telle que

$\mathrm{DSC}(\varphi', G) \leq \mathrm{DSC}(\varphi, G)$. La racine r vérifie évidemment $\varphi(r) = 1$, et on note T le sous-arbre de r contenant $\varphi^{-1}(n)$ (le sommet numéroté en dernier par φ). On note aussi r_T la racine de T. De notre affirmation initiale on déduit facilement que $\varphi(r_T) < n - |T| + 1$, ce qui signifie qu'au moins un sommet de $G \backslash T$ est numéroté parmi les sommets de T. La nouvelle numérotation φ' (voir figure 3.8) est construite de telle façon que :

1. l'ordre relatif entre les sommets de T n'est pas modifié.
2. l'ordre relatif entre les sommets de $G \backslash T$ n'est pas modifié.
3. les sommets de T sont numérotés de $n - |T| + 1$ à n.
4. par conséquent les sommets de $G \backslash T$ sont numérotés de 1 à $n - |T|$.

Ces trois règles permettent clairement de définir la construction d'une unique numérotation φ'. Cette numérotation est compatible avec la topologie de G car (r, r_T) est le seul arc reliant T et le reste de G. Or, $\varphi'(r) = 1 < \varphi'(r_T)$.

Pour démontrer que $\mathrm{DSC}(\varphi', G) \leq \mathrm{DSC}(\varphi, G)$, on considère un arc (u, v) tel que $u \neq r$. Le cas $u = r$ sera étudié dans la suite. Soit $\Delta(u, v) = (\varphi(v) - \varphi(u)) - (\varphi'(v) - \varphi'(u))$. $\Delta(u, v)$ ainsi défini représente la diminution de coût pour (u, v) causé par la transformation de φ en φ'. Par construction, $\Delta(u, v)$ est positif. En effet, si (u, v) est dans T, alors $\Delta(u, v)$ est égal au nombre de sommets $w \notin T$ tels que $\varphi(u) < \varphi(w) < \varphi(v)$. Symmétriquement, si (u, v) n'est pas dans T, alors $\Delta(u, v)$ est égal au nombre de sommets $w \in T$ tels que $\varphi(u) < \varphi(w) < \varphi(v)$. Par conséquent le coût d'accés de chaque sommet $u \neq r$ a diminué, c'est à dire qu'en terme de coût d'accés à la mémoire comme défini en 2.1, on a $\mathrm{DSC}(\varphi', u) \leq \mathrm{DSC}(\varphi, u)$.

Cependant, le coût d'accés $\mathrm{DSC}(\varphi, r)$ à la racine r de G augmente en général. La différence est égale à

$$\mathrm{DSC}(\varphi', r) - \mathrm{DSC}(\varphi, r) = \max_{(r,v) \in A} \varphi'(v) - \max_{(r,v) \in A} \varphi(v) \leq \varphi'(r_T) - \varphi(r_T)$$

On rappelle que $\Delta(r_T)$ représente la partie droite de l'inégalité. On observe de plus que $\Delta(r_T)$ est égal au nombre de sommets $w \notin T$ tels que $\varphi(w) > \varphi(r_T)$.

On peut conclure que $\mathrm{DSC}(\varphi', G) \leq \mathrm{DSC}(\varphi, G)$ en prouvant que la diminution totale des coûts des sommets $u \neq r$ est au moins égale à $\Delta(r_T)$. $\varphi^{-1}(T)$ est un sous ensemble de $\{1, \cdots, n\}$, et donc il peut être vu comme l'union d'intervalles d'entiers $I_1 < I_2 \cdots < I_k$. Clairement, φ est une bijection entre les sommets de T et $\bigcup_{i=1}^{k} I_i$. Considérons l'arc (u_1, v_1) de T tel que $\varphi(u_1) \in I_1$ et $\varphi(v_1)$ est maximal. Comme T est connexe, $\varphi(v_1)$ appartient à un certain intervalle I_{k_1} avec $k_1 > 1$. v_1 est alors par construction le dernier successeur de u_1 selon φ, donc le coût de u_1 est $\varphi(v_1) - \varphi(u_1)$. Si $k_1 < k$, on peut itérer ce processus de contruction : soit (u_2, v_2) de T tel que $\varphi(u_2) \in I_1$ et $\varphi(v_2)$ est maximal. Soit I_{k_2} l'intervalle qui contient $\varphi(v_2)$. A la fin, on obtient une séquence d'arcs $(u_1, v_1), \cdots, (u_l, v_l)$ et d'intervalles $I_1 = I_{k_0}, I_{k_2}, \cdots, I_{k_l} = I_k$ tels que $\varphi(u_i) \in I_{k_{i-1}}$ et $\varphi(v_i) \in I_{k_i}$. On a aussi

$\mathrm{DSC}(\varphi, u_i) = \varphi(v_i) - \varphi(u_i)$ et $\mathrm{DSC}(\varphi', u_i) = \varphi'(v_i) - \varphi'(u_i)$. Alors, $\mathrm{DSC}(\varphi', u_i) - \mathrm{DSC}(\varphi, u_i)$ est égal au nombre de sommets $w \notin T$ tels que $\varphi(u_i) < \varphi(w) < \varphi(v_i)$. Donc, puisque v_i et u_{i+1} sont dans le même intervalle l_{k_i}, la somme $\sum_{i=1}^{l}(\mathrm{DSC}(\varphi', u_i) - \mathrm{DSC}(\varphi, u_i))$ vaut $\Delta(r_T)$. Par conséquent, $\mathrm{DSC}(\varphi', G) \leq \mathrm{DSC}(\varphi, G)$, ce qui conclut la preuve. □

Il nous reste maintenant à déterminer comment sélectionner le sous-arbre terminal. Pour chaque successeur u de r, soit $T(u)$ le sous-arbre enraciné en u. Si $T(u^\star)$ représente le sous-arbre final, nous avons $\mathrm{DSC}(\varphi, r) = n - |T(u^\star)|$, et donc

$$\mathrm{DSC}(\varphi, G) = n - |T(u^\star)| + \sum_{u \in \Gamma^+(r)} \mathrm{DSC}(\varphi, T(u))$$

Par conséquent, pour pouvoir minimiser le coût DSC de la numérotation d'une arborescence G, $u^\star$ doit être choisi de façon à ce que $T(u^\star)$ soit le plus grand sous-arbre de r. La numérotation optimale sera donc donnée par l'algorithme récursif suivant, appelé avec la racine de G comme premier paramètre, et 1 comme second paramètre :

proc orderouttree(r,i) **:**
  $\varphi(r) \leftarrow i$
  si r n'est pas une feuille **alors**
    soit $u^\star$ un successeur de r tel que $T(u^\star) = \max_{(r,u)\in A} |T(u)|$
    pour chaque successeur $u \neq u^\star$ de r **faire**
      orderouttree(u,$i+1$)
      $i \leftarrow i + |T(u)|$
    finpour
    orderouttree($u^\star$, $i+1$)
  finsi

Pour conclure, nous montrons la complexité de cet algorithme.

Théorème 3.2.2. *Une numérotation optimale d'une arborescence au sens du critère* DSC *peut être obtenue en $\mathcal{O}(n)$ opérations.*

Preuve Dans la phase préalable, les tailles $|T(u)|$ des sous-arbres de tous les sommets u de G peuvent être calculées en temps $\mathcal{O}(n)$. La procédure récursive est appelée une fois pour chaque sommet, et le choix de $u^\star$ pour un sommet r donné se fait en $\mathcal{O}(\delta^+(r))$ opérations. Par conséquent le nombre total d'opérations pour l'algorithme est de $\mathcal{O}(n)$. □

Polynomialité de minUCS pour les arborescences

On suppose toujours que $G = (V, A)$ est une arborescence ($|A| = m = n - 1$). Nous prouvons dans ce qui suit que la numérotation calculée pour le score DSC en section 3.2.2,

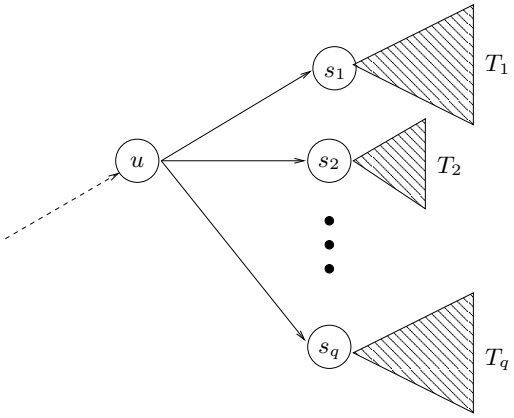


FIG. 3.9 – Un sommet u d'une arborescence et ses successeurs

notée φ^*, est aussi optimale pour le score UCS. Nous commençons par prouver une inégalité reliant les 2 scores.

Lemme 6. *Dans une arborescence G,* $\mathrm{UCS}(\varphi) \geq \mathrm{DSC}(\varphi) - m$ *pour toute numérotation φ.*

Preuve Considérons l'unique opération LD associée à un arc (u, v) de G. Nous allons prouver que $\mathrm{UCS}(\varphi, (u, v)) \geq \mathrm{DSC}(\varphi, (u, v)) - 1$. Si u est déjà au sommet de la pile, cela signifie qu'il s'agit du résultat d'une opération RD ou OP effectuée juste avant. Par conséquent dans ce cas on a $\varphi(v) = \varphi(u) + 1$, et v est le premier successeur de u. Dans ce cas, $\mathrm{UCS}(\varphi, (u, v)) = 0 = \mathrm{DSC}(\varphi, (u, v)) - 1$. Sinon (si u n'est pas au sommet de la pile) alors v est le $i^{\text{ème}}$ successeur de u pour un certain entier i vérifiant $1 \leq i \leq \delta^+(u)$. La dernière fois que u était au sommet de la pile était quand u a été créé (si $i = 1$) ou juste avant le calcul de $s_{i-1}(u)$ (si $i > 1$). Dans les deux cas, au moins $\varphi(s_i(u)) - \varphi(s_{i-1}(u)) - 1$ variables ont été empilées par des opérations OP depuis. Donc, $\mathrm{UCS}(\varphi, (u, v)) \geq \varphi(s_i(u)) - \varphi(s_{i-1}(u)) - 1 = \mathrm{DSC}(\varphi, (u, v)) - 1$. La sommation de toutes ces inégalités conduit au résultat du lemme. □

Théorème 3.2.3. *Pour une arborescence G, la solution optimale $\varphi^\star$ au sens de* DSC *est aussi optimale au sens de* UCS*. Le problème peut donc être aussi résolu en $\mathcal{O}(n)$ opérations.*

Preuve Nous prouvons dans la suite que $\mathrm{UCS}(\varphi^\star, G) = \mathrm{DSC}(\varphi^\star, G) - m$. Ceci nous permettrait de conclure car dans ce cas comme on sait que $\varphi^\star$ est optimale pour DSC, et que $\mathrm{DSC}(\varphi^\star, G) - m$ est une borne inférieure pour UCS (lemme 6), nous conclurons que $\varphi^\star$ est aussi optimale pour UCS.

On considère, comme illustré en figure 3.9, un sommet u et ses $q = \delta^+(u)$ successeurs, notés $s_1 = s_1(\varphi^\star, u), \cdots, s_q = s_q(\varphi^\star, u)$. Donc, $\varphi^\star(s_1) < \cdots < \varphi^\star(s_q)$. On définit $T_i, i \in \{1, \cdots, n\}$, les sommets des sous-arbres dont la racine est s_i. A chaque étape

$i \in \{1, \cdots, q\}$, u est chargé pour le calcul de s_i. V_i représente alors l'ensemble des sommets empilés entre u et le sommet de la pile juste avant l'évaluation de s_i. Le coût UCS de ce chargement de u peut alors être exprimé comme $\mathrm{UCS}(\varphi^\star, (u, s_i)) = |V_i|$. Deux cas doivent être étudiés :

- Si $i = 1$, par définition de $\varphi^\star$, $\varphi^\star(s_1) = \varphi^\star(u) + 1$. u est déjà au sommet de la pile, donc $V_1 = \emptyset$. On obtient alors $|V_1| = 0 = \varphi^\star(s_1) - \varphi^\star(u) - 1$.
- Sinon, c'est à dire si $i > 1$, on prouve que $V_i = T_{i-1}$:
  - Par définition de $\varphi^\star$, les éléments de T_{i-1} sont calculés après le calcul de s_{i-1} et avant le chargement de u pour le calcul de s_i, donc $T_{i-1} \subset V_i$.
  - Réciproquement, soit $k \in V_i \backslash T_{i-1}$. $\varphi^\star(k) < \varphi^\star(s_{i-1})$ car les éléments de T_{i-1} sont numérotés de $\varphi^\star(s_{i-1})$ à $\varphi^\star(s_i) - 1$. Par conséquent, il existe un sommet $l \in T_{i-1}$ dont le calcul nécessite le chargement de k car par construction tous les éléments numérotés entre s_{i-1} et s_i sont dans T_{i-1}. puisque $k \notin T_{i-1}$, l a deux prédécesseurs, ce qui contredit la structure de G. Donc $T_{i-1} = V_i$.

  Par conséquent,

$$\mathrm{UCS}(\varphi^\star, (u, s_i)) = |T_{i-1}| = \varphi^\star(s_i) - \varphi^\star(s_{i-1}) - 1 = \mathrm{DSC}(\varphi^\star, (u, s_i)) - 1$$

En additionant tous ces coûts, on obtient que $\mathrm{UCS}(\varphi, G) = \mathrm{DSC}(\varphi, G) - m$, ce qui complète la preuve. □

3.2.3 Graphes série-parallèles

L'objectif de cette partie est de montrer une autre classe de graphes définis récursivement pour laquelle minDSC peut être résolu en temps polynomial : une classe de graphes série-parallèles. L'intérêt de l'étude des graphes série-parallèles dans le cadre de nos travaux est qu'ils sont souvent utilisés dans l'analyse statique de programmes. Il existe en fait beaucoup de définitions des graphes série-parallèles. Les plus répandues sont la classe 2TSPG (2-terminal series parallel graphs) [Sch95, VTL79], et la classe SP [Bru95]. Nous montrons cependant à l'aide de contre-exemples en 3.2.3 que l'exploitation de la structure récursive de ces graphes ne peut mener à des numérotations optimales en temps polynomial. C'est pourquoi nous nous intéressons ici à une classe de graphes série-parallèles introduite par [CP95] pour l'étude de problèmes d'ordonnancement avec délais. Il s'agit de la classe r-2TSPG (pour **reduced 2-terminal series parallel graphs**). Elle a la propriété d'être incluse dans 2TSP, mais la limitation du nombre d'arêtes reliant un composant à un composant de niveau hiérarchique différent nous a permis de simplifier la formulation du coût DSC.

Tout d'abord, nous définissons les différentes classes en jeu dans la section 3.2.3. Nous montrons aussi la relation entre ces classes. Ensuite, dans la partie 3.2.3, nous montrons la dominance des numérotations par blocs pour le coût DSC dans la classe r-2TSPG. Ensuite, nous prouvons qu'une numérotation optimale peut être atteinte en temps poly-

nomial. Finalement, nous concluons (partie 3.2.3) en montrant par des contre-exemples que l'algorithme ne s'applique pas dans deux autres classes de graphes série-parallèles (2TSPG et SP).

Définitions et exemples

L'objectif de cette partie est double. Tout d'abord, les définitions des trois classes de graphes série-parallèles qui nous intéressent sont données. Ensuite, on montre que r-2TSPG est incluse dans 2TSP.

Définition 4 (Graphe r-2TSPG ([CP95])). *Le graphe composé de 2 sommets s et p connectés par un seul arc (s, p) est le graphe de base pour la classe r-2TSPG. Les graphes composés peuvent être obtenus à partir de K graphes $G_i, 1 \leq i \leq K$, de sommets source et puits respectifs s_i et p_i selon 2 règles de composition :*

Série *Fusionner p_i avec $s_{i+1}, \forall i, 1 \leq i \leq K-1$. Les source et puits de G sont respectivement s_1 and p_K (figure 3.11(a)),*

Parallèle *Créer la source et le puits de G s et p. Créer les arcs (s, s_i) et (p_i, p) $\forall i, 1 \leq i \leq K$ (figure 3.10(a)).*

Définition 5 (Graphe 2TSP ([Sch95, VTL79])). *Le graphe composé de 2 sommets s et p connectés par un seul arc (s, p) est le graphe de base pour la classe 2TSPG. Les graphes composés peuvent être obtenus à partir de K graphes $G_i, 1 \leq i \leq K$, de sommets source et puits respectifs s_i et p_i selon 2 règles de composition :*

Série *Fusionner p_i avec $s_{i+1}, \forall i, 1 \leq i \leq K-1$. Les source et puits de G sont respectivement s_1 and p_K (figure 3.11(a)),*

Parallèle *(figure 3.10(b))*

- *Fusionner tous les sommets sources $s_i, \forall i, 1 \leq i \leq K$ pour former s, le sommet source de G,*
- *Fusionner tous les sommets puits $p_i, \forall i, 1 \leq i \leq K$ pour former p, le sommet puits de G.*

Définition 6 (Graphe SP ([Bru95])). *Le graphe composé d'un sommet s est le graphe de base pour la classe SP. Les graphes composés peuvent être obtenus à partir de K graphes $G_i = (V_i, A_i), 1 \leq i \leq K$, selon 2 règles de composition (pour $i \in \{1, \cdots, K\}$, on note S_i (resp P_i) l'ensemble des sommets sans prédécesseurs (resp successeurs) de G_i) :*

Série *Le graphe SP résultant de la composition série est*

$$G = (\bigcup_{i=1\cdots K} V_i, (\bigcup_{i=1\cdots K} A_i) \cup (\bigcup_{i=1\cdots K-1} P_i \times S_{i+1})),$$

où "$\times$" représente l'opération de produit cartésien (figure 3.11(b)).

Parallèle *Le graphe SP résultant de la composition parallèle est $G = (\bigcup_{i=1\cdots K} V_i, \bigcup_{i=1\ldots K} A_i)$ (figure 3.10(c)).*

	Série	Parallèle
r-2TSPG	$\rightarrow_{\mathrm{r}}$	$\|_{\mathrm{r}}$
2TSP	$\rightarrow_{\mathrm{2TSP}}$	$\|_{\mathrm{2TSP}}$
SP	$\rightarrow_{\mathrm{SP}}$	$\|_{\mathrm{SP}}$

TAB. 3.1 – Notations pour les compositions des classes de graphes série-parallèles

Afin de clarifier la suite, les notations utilisées pour décrire les six opérations de composition sont décrites dans le tableau 3.1.

Un exemple de graphe de 2TSP (resp. SP) est montré en figure 3.22 (resp. figure 3.23). Nous montrons maintenant que les graphes de r-2TSPG sont aussi dans 2TSP.

Propriété 2.

$$\textit{r-2TSPG} \subset \textit{2TSP}$$

Preuve *(par récurrence sur les opérations de composition de graphes)* Le graphe de base de r-2TSPG, noté B, appartient à 2TSP. Soient deux graphes G_1 et G_2 appartenant à r-2TSPG. On suppose qu'ils appartiennent aussi à 2TSP (hypothèse de récurrence). On veut montrer (i) que $(G_1 \rightarrow_{\mathrm{r}} G_2) \in$ 2TSP et (ii) que $(G_1 \|_{\mathrm{r}} G_2) \in$ 2TSP.

(i) Les compositions série sont identiques, donc la réponse à la première question est triviale,

(ii) Le graphe $(G_1 \|_{\mathrm{r}} G_2)$ peut être vu comme la composition parallèle au sens de 2TSP de deux graphes utilisant la composition série au sens de 2TSP :

$$(G_1 \|_{\mathrm{r}} G_2) = ((B \rightarrow_{\mathrm{2TSP}} G_1 \rightarrow_{\mathrm{2TSP}} B) \|_{\mathrm{2TSP}} (B \rightarrow_{\mathrm{2TSP}} G_2 \rightarrow_{\mathrm{2TSP}} B))$$

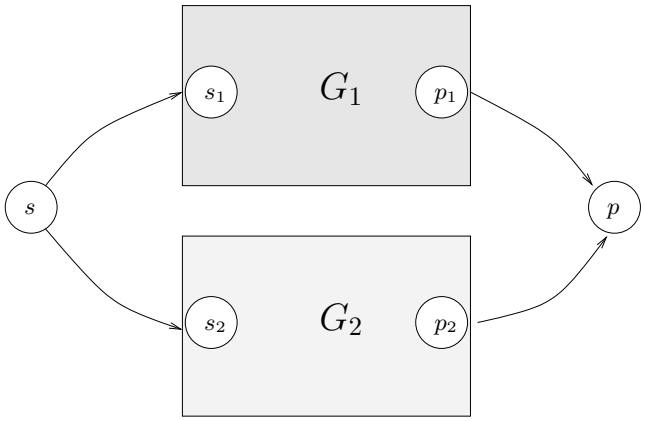
Or, $(B \rightarrow_{\mathrm{2TSP}} G_1 \rightarrow_{\mathrm{2TSP}} B) \in$ 2TSP et $(B \rightarrow_{\mathrm{2TSP}} G_2 \rightarrow_{\mathrm{2TSP}} B) \in$ 2TSP, puisqu'on a supposé par récurrence que $G_1 \in$ 2TSP et $G_2 \in$ 2TSP. D'où le résultat.

Le cas des composition parallèles n-aires est similaire. L'inclusion est stricte, comme le montre l'exemple de la figure 3.22. □

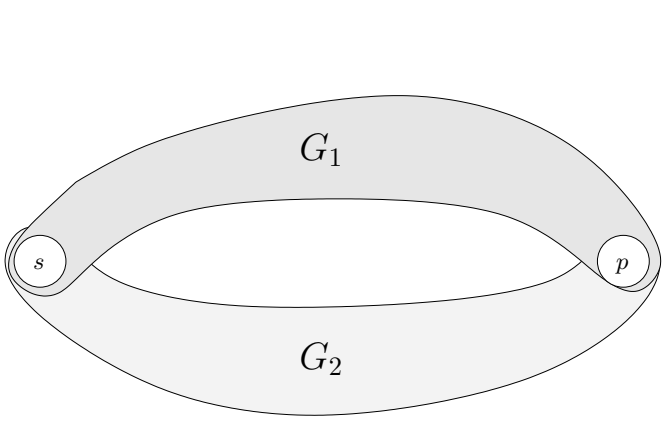
Nous allons maintenant montrer que la structure des graphes de r-2TSPG conduit à des numérotations dominantes pour le problème minDSC.

Dominance des numérotations par blocs

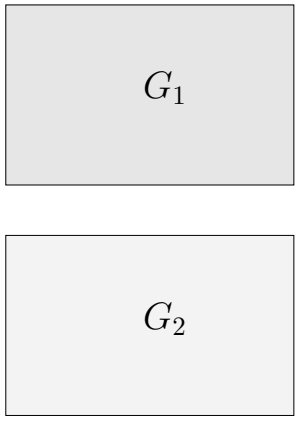
Soit $G = (V, A)$ le graphe de r-2TSPG obtenu par composition parallèle de K graphes de r-2TSPG, notés $G_i = (V_i, A_i), 1 \leq i \leq K$. Soient s et p la source et le puits de G, et on note respectivement s_i et p_i les sommets source et puits des graphes G_i, $i \in \{1, \cdots, K\}$. Soit $\varphi = \varphi^{(0)}$ une numérotation de G.



(a) Composition r-2TSPG

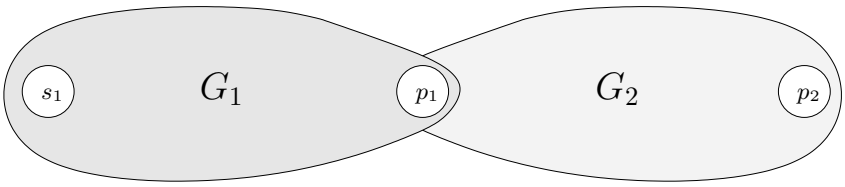


(b) Composition 2TSP

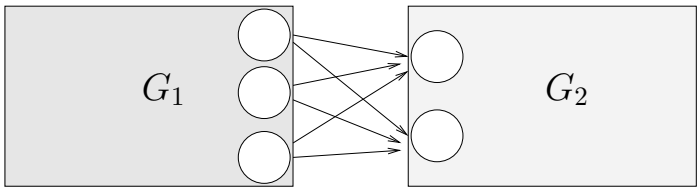


(c) Composition SP

Fig. 3.10 – Les différents types de composition parallèle



(a) Composition r-2TSPG et 2TSP



(b) Composition SP

FIG. 3.11 – Les différents types de composition série

Pour tout $i \in \{1, \cdots, K\}$, on note k_i le nombre d'ensembles dans lesquels on a coupé G_i :

$$\forall i \in \{1, \cdots, K\}, k_i := |\{u \in V_i, \varphi^{-1}(\varphi(u)+1) \notin V_i\}|$$

Pour tout $i \in \{1, \cdots, K\}$ et pour tout $j \in \{1, \cdots, k_i\}$, on note B_i^j le $j^{\text{ème}}$ ensemble de sommets consécutifs de G_i selon φ.

Un exemple de ces notations est montré en figure 3.13. Finalement, on note Ω_φ l'ensemble des arcs qui portent un coup au sens de DSC :

$$\Omega_\varphi = \{(z,t) \in G, \varphi(t) = \max_{v \in \Gamma^+(z)} \varphi(v)\}$$

Pour tout $z \in V - \{p\}$, le sommet $t \in V$ tel que $(z,t) \in \Omega_\varphi$ est appelé le **dernier successeur** de z.

Définition 7 (numérotation par blocs). *φ est une* **numérotation par blocs** *si* $\forall i \in \{1, \cdots, K\}, \forall (x,y) \in V_i \times V_i, \forall z \in V$ *tel que* $\varphi(x) < \varphi(z) < \varphi(y)$, *alors* $z \in V_i$.

Dans le cas de numérotations par blocs, on considèrera aussi un autre type de numérotation appelée **fonction d'étiquetage de blocs**, qui consiste à numéroter des blocs au lieu de sommets. Une définition plus formelle en est donnée en 3.2.3.

Pour prouver notre méthode, nous n'étudierons que les compositions parallèles, car les compositions en série sont évidemment triviales en termes de score DSC pour des

raisons de construction. Nous montrons ici que pour toute numérotation, il est possible de construire une numérotation par blocs de coût moindre. Cette numérotation par blocs est construite par concaténations successives d'ensembles de sommets appartenant au même sous-graphe. L'étape $i, i \in \{1, \cdots, K-1\}$ consiste à concaténer les ensembles $B_i^j, j \in \{1, \cdots, k_i\}$, avec B_i^1, où i est l'indice minimal des sous-graphes qui n'ont pas encore été concaténés (voir figure 3.12). Pour tout $i_0 \in \{1, \cdots, K\}$ tel que $k_{i_0} = 1$, on considère simplement que $\varphi^{(i_0)} = \varphi^{(i_0-1)}$. De cette façon, on ne considère dans ce qui suit que les étapes i vérifiant $k(i) > 1$.

Bien évidemment, $\varphi^{(K-1)}$ est une numérotation par blocs.

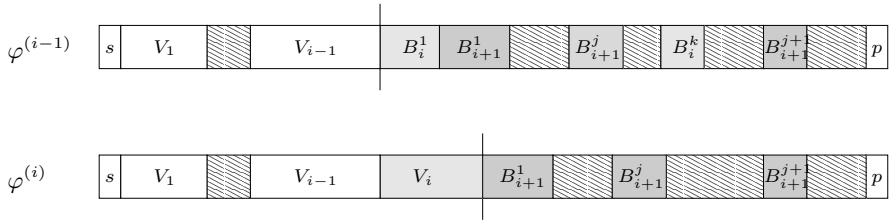


FIG. 3.12 – Transformation de $\varphi^{(i-1)}$ en $\varphi^{(i)}$

$$\begin{aligned}\Omega_{\varphi^{(0)}} &= \{(s, s_3), (s_1, b), (s_2, c), (s_3, d), (a, p_1),\\ &\quad (b, p_1), (c, p_2), (d, p_3), (p_1, p), (p_2, p), (p_3, p)\}\end{aligned}$$

$$k_1 = k_2 = 3, k_3 = 2$$

j	1	2	3
Θ_1^j	s_1	a, b	p_1
Θ_2^j	s_2	c	p_2
Θ_3^j	s_3	p_3	non défini

TAB. 3.2 – Valeurs pour le graphe de la figure 3.13

Nous allons montrer dans ce qui suit que :

$$\forall i \in \{1, \cdots, K-1\}, \mathrm{DSC}(\varphi^{(i)}, G) - \mathrm{DSC}(\varphi^{(i-1)}, G) \leq 0$$

Puisque $\mathrm{DSC}(\varphi^{(i)}, G_l) = \mathrm{DSC}(\varphi^{(i-1)}, G_l), \forall l \in \{1, \cdots, i-1\}$, nous pouvons écrire :

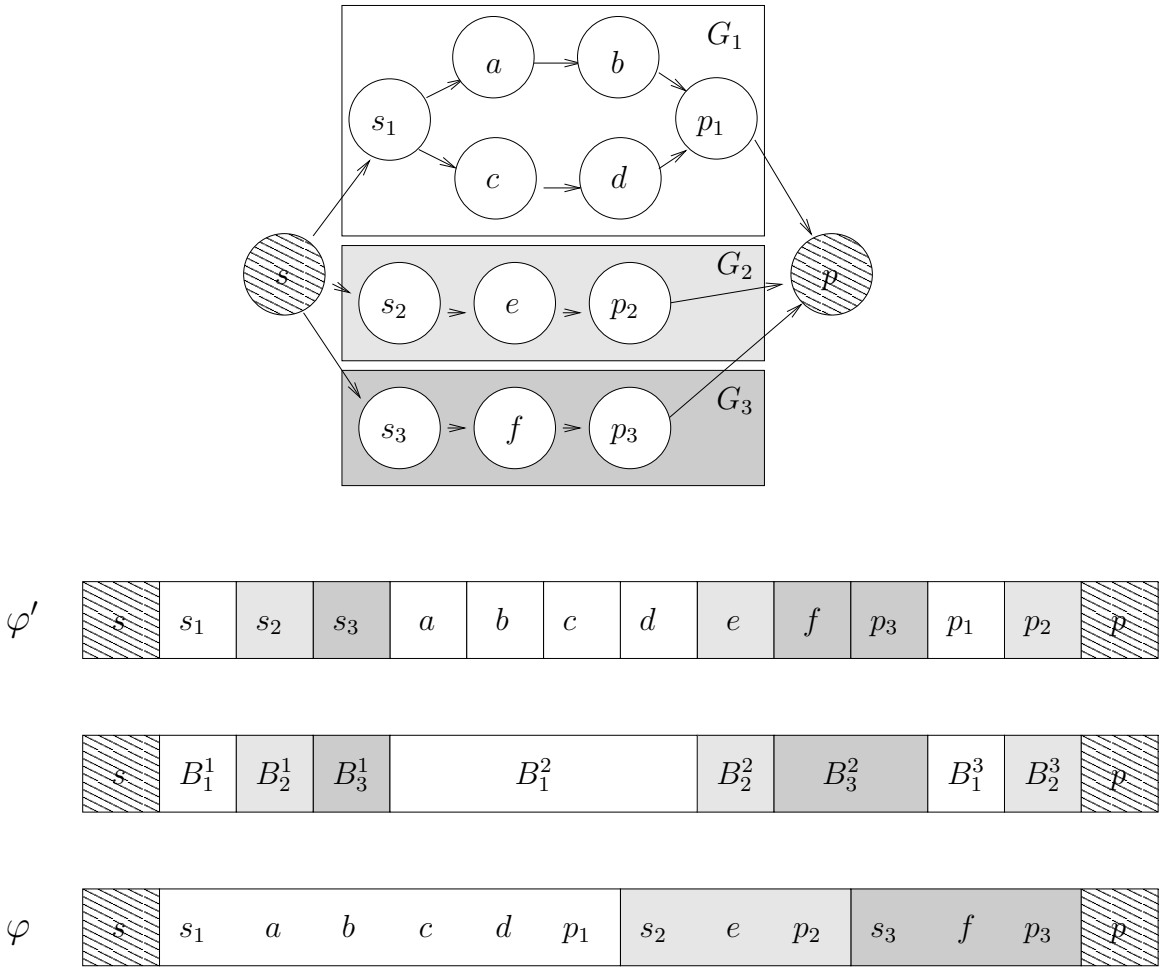


FIG. 3.13 – Exemple de construction d'une numérotation par blocs

$$\begin{aligned}\mathrm{DSC}(\varphi^{(i)},G)-\mathrm{DSC}(\varphi^{(i-1)},G) &= \mathrm{DSC}(\varphi^{(i)},s)-\mathrm{DSC}(\varphi^{(i-1)},s)\\ &+ \mathrm{DSC}(\varphi^{(i)},G_i)-\mathrm{DSC}(\varphi^{(i-1)},G_i)\\ &+ \mathrm{DSC}(\varphi^{(i)},G_{i+1})-\mathrm{DSC}(\varphi^{(i-1)},G_{i+1})\\ &+ \sum_{l=i+2}^{K}\mathrm{DSC}(\varphi^{(i)},G_l)-\mathrm{DSC}(\varphi^{(i-1)},G_l)\end{aligned}$$

Les lemmes qui suivent vont nous permettre de montrer une borne pour chacun des termes, de façon à conclure (voir théorème 3.2.4).

Lemme 7. $\forall i \in \{1,\cdots,K-1\}, \mathrm{DSC}(\varphi^{(i)},s)-\mathrm{DSC}(\varphi^{(i-1)},s) \leq \varphi^{(i)}(s_{i+1})-\varphi^{(i-1)}(s_{i+1})$

Preuve Le dernier successeur de s pour $\varphi^{(i)}$ et $\varphi^{(i-1)}$ est s_K. Par conséquent,

$$\mathrm{DSC}(\varphi^{(i)},s)-\mathrm{DSC}(\varphi^{(i-1)},s)=\varphi^{(i)}(s_K)-\varphi^{(i-1)}(s_K)$$

De $\varphi^{(i-1)}$ à $\varphi^{(i)}$, le nombre de sommets de V déplacés entre s_i et s_K est au plus $|V_i| - |B_i^1|$. Par conséquent,

$$|V_i| - |B_i^1| \geq \varphi^{(i)}(s_K) - \varphi^{(i-1)}(s_K)$$

Maintenant, $\varphi^{(i)}(s_{i+1}) = \varphi^{(i)}(s_i) + |V_i|$ et $\varphi^{(i-1)}(s_{i+1}) = \varphi^{(i-1)}(s_i) + |B_i^1| = \varphi^{(i)}(s_i) + |B_i^1|$. En sommant les deux équations on obtient $|V_i| - |B_i^1| = \varphi^{(i)}(s_{i+1}) - \varphi^{(i-1)}(s_{i+1})$, ce qui nous permet de conclure. □

Lemme 8. $\forall i \in \{1, \cdots, K-1\}, \sum_{l=i+2}^{K} \big(\mathrm{DSC}(\varphi^{(i)}, G_l) - \mathrm{DSC}(\varphi^{(i-1)}, G_l)\big) \leq 0$

Preuve Soit $(v, w) \in \Omega_{\varphi^{(i)}}$ avec $(v, w) \in V_l \times (V_l \cup \{p\}), l \in \{i+2, \cdots, K\}$.

(a) Si il existe $\beta \in \{1, \cdots, k_l\}$ tel que v et w appartiennent à un même sous ensemble B_l^β, alors :

$$\begin{aligned} \mathrm{DSC}(\varphi^{(i-1)}, v) &= \varphi^{(i-1)}(w) - \varphi^{(i-1)}(v) \\ &= \varphi^{(i)}(w) - \varphi^{(i)}(v) = \mathrm{DSC}(\varphi^{(i)}, v) \end{aligned}$$

(b) Sinon, soit $\beta_1, \beta_2 \in \{1, \cdots, k_l\}^2$ avec $v \in B_l^{\beta_1}$ et $w \in B_l^{\beta_2}$. Alors, tous les sommets des sous ensembles B_i^j numérotés par $\varphi^{(i-1)}$ entre $B_l^{\beta_1}$ et $B_l^{\beta_2}$ sont numérotés par $\varphi^{(i)}$ avant B_l^1.
Donc,

$$\mathrm{DSC}(\varphi^{(i-1)}, v) = \varphi^{(i-1)}(w) - \varphi^{(i-1)}(v) \geq \varphi^{(i)}(w) - \varphi^{(i)}(v) = \mathrm{DSC}(\varphi^{(i)}, v)$$

On en déduit que pour tout $l \in \{i+2, \cdots, K\}$

$$\mathrm{DSC}(\varphi^{(i)}, G_l) - \mathrm{DSC}(\varphi^{(i-1)}, G_l) \leq 0$$

□

On définit pour tout $i \in \{1, \cdots, K\}$ et pour tout $j \in \{1, \cdots, k_i - 1\}$ (voir le tableau 3.2 et la figure 3.14 pour un exemple)

$$\Theta_i^j(\varphi) = \{z, (z,t) \in \Omega_\varphi, z \in B_i^j \text{ et } t \in \bigcup_{l=j+1}^{k_i} B_i^l\}$$

$\Theta_i^j(\varphi)$ est l'ensemble des sommets de B_i^j dont le dernier successeur n'est pas dans B_i^j. On définit aussi

$$\Theta_i^{k_i}(\varphi) = \{p_i\} = \{z, (z,t) \in \Omega_\varphi, z \in B_i^{k_i} \text{ et } t = p\}$$

Finalement, pour tout $i \in \{1, \cdots, K\}$ et pour tout $j \in \{1, \cdots, k_i\}$, on définit $\theta_i^j(\varphi) = |\Theta_i^j(\varphi)|$.

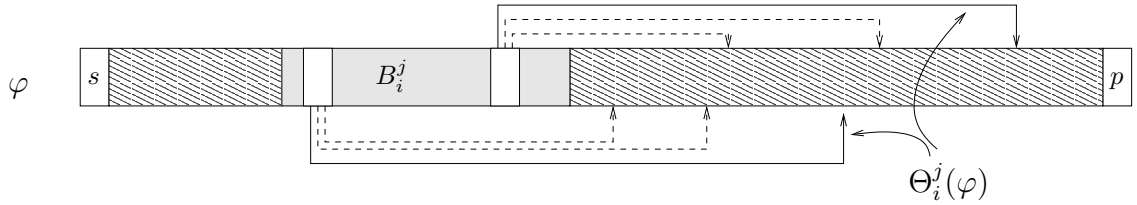


FIG. 3.14 – Interprétation de $\Theta_i^j(\varphi)$

Lemme 9. *Soit φ une numérotation de G. Pour tout $i \in \{1, \cdots, K\}$ et pour tout $j \in \{1, \cdots, k_i\}$, $\theta_i^j(\varphi) > 0$*

Preuve
(a) $\theta_i^{k_i}(\varphi) = |\Theta_i^{k_i}(\varphi)| = 1$.
(b) Supposons qu'il existe $j_0 \in \{1, \cdots, k_i - 1\}$ tel que $\theta_i^{j_0}(\varphi) = 0$.
- S'il n'existe pas d'arc $(u, v) \in A_i, u \in B_i^{j_0}$ et $v \in \bigcup_{l=j_0+1}^{k_i} B_i^l$ alors il n'existe pas de chemin entre un élément $u \in B_i^{j_0}$ et p_i, donc G_i n'est pas connexe.
- S'il n'existe pas d'arc $(u, v) \in \Omega_\varphi, u \in B_i^{j_0}$ et $v \in \bigcup_{l=j_0+1}^{k_i} B_i^l$, alors le dernier successeur w de u est numéroté après v avec $w \in \bigcup_{l=j_0+1}^{k_i} B_i^l$. Donc, $(u, w) \in \Omega_\varphi$.

□

Pour $i \in \{1, \cdots, K-1\}$ and $l \in \{i, \cdots, K\}$, nous définissons une suite strictement croissante $u_\beta \in \{1, \cdots, k_l\}$ de $\alpha > 1$ éléments comme suit :

$$\begin{cases} u_1 = 1 \\ \forall \beta \in \{2, \cdots, \alpha\}, u_\beta \in \{u_{\beta-1} + 1, \cdots, k_l\} \text{ tel que} \\ \quad \exists (v, w) \in \Omega_{\varphi^{(i-1)}} \text{ avec } v \in B_l^{u_{\beta-1}}, w \in B_l^{u_\beta} \\ u_\alpha = k_l \end{cases}$$

Soient $(v_\beta), \beta \in \{1, \cdots, \alpha\}$ et $(w_\beta), \beta \in \{2, \cdots, \alpha + 1\}$ des suites de sommets de $V_l \cup \{p\}$ telles que :
- Pour tout $\beta \in \{1, \cdots, \alpha - 1\}$, $(v_\beta, w_{\beta+1}) \in \Omega_{\varphi^{(i-1)}}$ avec $v_\beta \in B_l^{u_\beta}, w_{\beta+1} \in B_l^{u_{\beta+1}}$
- $v_\alpha = p_i$, $w_{\alpha+1} = p$.

Selon le lemme 9, $\theta_l^k(\varphi^{(i-1)}) > 0$ pour tout $k \in \{1, \cdots, k_l\}$, donc les suites (u_β), (v_β) et (w_β) existent. Voir la figure 3.15 pour un exemple.

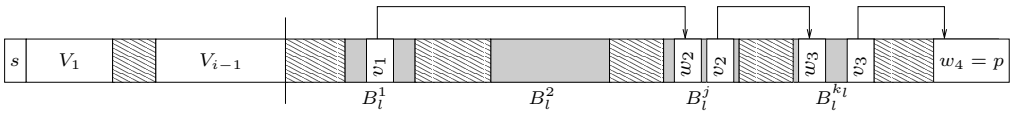


FIG. 3.15 – Exemple de suites pour $\varphi^{(i-1)}$

Pour plus de clarté, nous utilisons la notation suivante (voir figure 3.16) :

$$\forall i \in \{1, \cdots, K-1\}, \forall j \in \{1, \cdots, k_i\}, h_i^j := \text{ le premier sommet de } B_i^j$$

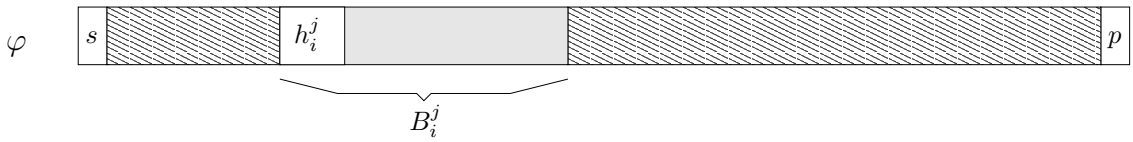


FIG. 3.16 – Illustration pour h_i^j

Lemme 10. $\forall i \in \{1, \cdots, K-1\}, \mathrm{DSC}(\varphi^{(i)}, G_i) - \mathrm{DSC}(\varphi^{(i-1)}, G_i) \leq 0$

Preuve Soit $i \in \{1, \cdots, K-1\}$

$$\mathrm{DSC}(\varphi^{(i)}, G_i) - \mathrm{DSC}(\varphi^{(i-1)}, G_i) = \sum_{v \in V_i} \big(\mathrm{DSC}(\varphi^{(i)}, v_i) - \mathrm{DSC}(\varphi^{(i-1)}, v_i)\big)$$

(a) $\mathrm{DSC}(\varphi^{(i-1)}, p_i) = \varphi^{(i-1)}(p) - \varphi^{(i-1)}(p_i) \geq 0$

$\mathrm{DSC}(\varphi^{(i)}, p_i) = \varphi^{(i)}(p) - \varphi^{(i)}(p_i) = \sum_{l=i+1}^{K} |V_l|$ puisque les sommets numérotés entre $\varphi^{(i)}(p_i)+1$ et $\varphi^{(i)}(p)-1$ dans $\varphi^{(i)}$ sont ceux de $\bigcup_{l=i+1}^{K} V_l$.

Donc, $\mathrm{DSC}(\varphi^{(i)}, p_i) - \mathrm{DSC}(\varphi^{(i-1)}, p_i) \leq \sum_{l=i+1}^{K} |V_l|$.

(b) Soit $(u_\beta), (v_\beta)$ et (w_β) les suites définies précédemment, avec $l = i$. On définit $V(i) = \{v_\beta, \beta \in \{1, \cdots, \alpha\}\}$. On définit aussi les ensembles suivants $\forall \beta \in \{1, \cdots, \alpha-1\}, \forall p \in \{i+1, \cdots, K\}$:

$$\begin{aligned} L_{\beta,p} &= \{m \in \{1, \cdots, k_p\}, \varphi^{(i-1)}(h_i^{u_\beta}) < \varphi^{(i-1)}(h_p^m) < \varphi^{(i-1)}(h_i^{u_{\beta+1}})\} \\ L_{\alpha,p} &= \{m \in \{1, \cdots, k_p\}, \varphi^{(i-1)}(h_i^{u_\alpha}) < \varphi^{(i-1)}(h_p^m)\} \end{aligned}$$

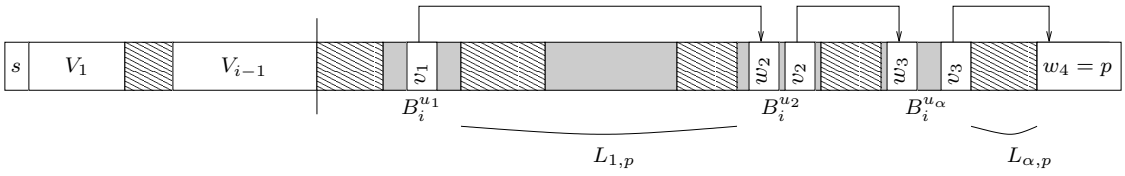


FIG. 3.17 – Exemple pour le lemme 10

Pour tout $v_\beta \in V(i)$, on a :

$$\mathrm{DSC}(\varphi^{(i)}, v_\beta) - \mathrm{DSC}(\varphi^{(i-1)}, v_\beta) = -\sum_{p=i+1}^{K} \sum_{m \in L_{\beta,p}} |B_p^m|$$

Donc :

$$\sum_{v_\beta \in V(i)} \mathrm{DSC}(\varphi^{(i)}, v_\beta) - \mathrm{DSC}(\varphi^{(i-1)}, v_\beta) = - \sum_{p=i+1}^{K} \sum_{\beta=1}^{\alpha} \sum_{m \in L_{\beta,p}} |B_p^m|$$

Maintenant,

$$\bigcup_{\beta=1}^{\alpha} L_{\beta,p} = \{m \in \{1, \cdots, k_p\}, \varphi^{(i-1)}(h_i^1) < \varphi^{(i-1)}(h_p^m)\}$$

Puisque tous les sommets de $V_p, p \in \{i+1, \cdots, K\}$ sont numérotés par $\varphi^{(i-1)}$ après B_i^1, on a $\bigcup_{\beta=1}^{\alpha} L_{\beta,p} = \{1, \cdots, k_p\}$ et :

$$\begin{aligned} \sum_{v_\beta \in V(i)} \mathrm{DSC}(\varphi^{(i)}, v_\beta) - \mathrm{DSC}(\varphi^{(i-1)}, v_\beta) &= - \sum_{p=i+1}^{K} \sum_{m=1}^{k_p} |B_p^m| \\ &= - \sum_{p=i+1}^{K} |V_p| \end{aligned}$$

(c) Finalement, pour tout sommet $v \in V_i - \{p_i\} - V(i)$ et pour tout $w \in \Gamma^+(v), \varphi^{(i)}(w) - \varphi^{(i)}(v) \leq \varphi^{(i-1)}(w) - \varphi^{(i-1)}(v)$. Par conséquent :

$$\mathrm{DSC}(\varphi^{(i)}, v) - \mathrm{DSC}(\varphi^{(i-1)}, v) \leq 0$$

Ce qui nous amène au résultat escompté. □

Lemme 11. *Pour tout $i \in \{1, \cdots, K-1\}$ et pour tout $l \in \{i+1, \cdots, K\}$, on a*

$$\mathrm{DSC}(\varphi^{(i-1)}, G_l) - \mathrm{DSC}(\varphi^{(i)}, G_l) \geq \sum_{j \in L_l} |B_i^j|,$$

où L_l est l'ensemble des indices $j \in \{2, \cdots, k_i\}$ tels que l'ensemble B_i^j est numéroté dans $\varphi^{(i-1)}$ après B_l^1 :

$$L_l = \{j \in \{2, \cdots, k_i\}, \varphi^{(i-1)}(h_l^1) < \varphi^{(i-1)}(h_i^j)\}$$

Preuve Soient $(u_\beta), (v_\beta)$ et (w_β) les séquences définies précédemment, pour une valeur donnée $l \in \{i+1, \cdots, K\}$ et pour la numérotation $\varphi^{(i-1)}$. On définit $V(l) = \{v_\beta, \beta \in \{1, \cdots, \alpha\}\}$. On définit aussi (voir figure 3.18) :

$$\forall \beta \in \{1, \cdots, \alpha - 1\}, L'_\beta = \{j \in \{2, \cdots, k_i\}, \varphi^{(i-1)}(h_l^{u_\beta}) < \varphi^{(i-1)}(h_i^j) < \varphi^{(i-1)}(h_l^{u_{\beta+1}})\}$$

$$L'_\alpha = \{j \in \{2, \cdots, k_i\}, \varphi^{(i-1)}(h_l^{u_\alpha}) < \varphi^{(i-1)}(h_i^j)\}$$

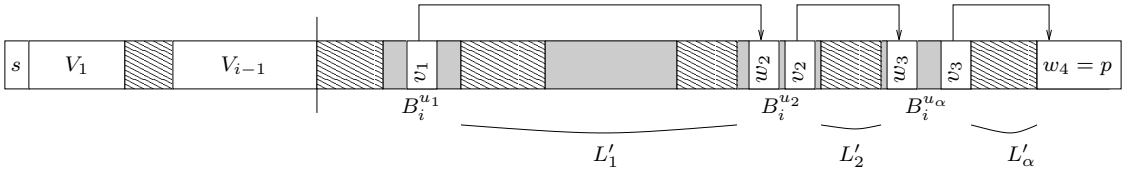


FIG. 3.18 – Exemple pour le lemme 11

(a) $\forall \beta \in \{1, \cdots, \alpha\}, \mathrm{DSC}(\varphi^{(i-1)}, v_\beta) - \mathrm{DSC}(\varphi^{(i)}, v_\beta) = \sum_{j \in L'_\beta} |B_i^j|$.

Puisque $L_l = \bigcup_{\beta=1}^{\alpha} L'_\beta$, on a :

$$\begin{aligned} \sum_{\beta=1}^{\alpha} \big(\mathrm{DSC}(\varphi^{(i-1)}, v_\beta) - \mathrm{DSC}(\varphi^{(i)}, v_\beta)\big) &= \sum_{\beta=1}^{\alpha} \sum_{j \in L'_\beta} |B_i^j| \\ &= \sum_{j \in L_l} |B_i^j| \end{aligned}$$

(b) Finalement, pour tout sommet $v \in V_l - V(l)$ et pour tout $w \in \Gamma^+(v), \varphi^{(i)}(w) - \varphi^{(i)}(v) \leq \varphi^{(i-1)}(w) - \varphi^{(i-1)}(v)$. Par conséquent :

$$\mathrm{DSC}(\varphi^{(i)}, v) - \mathrm{DSC}(\varphi^{(i-1)}, v) \leq 0$$

Ce qui nous permet de conclure.

□

Lemme 12. *Pour tout* $i \in \{1, \cdots, K-1\}$,

$$\mathrm{DSC}(\varphi^{(i)}, G_{i+1}) - \mathrm{DSC}(\varphi^{(i-1)}, G_{i+1}) \leq \varphi^{(i-1)}(s_{i+1}) - \varphi^{(i)}(s_{i+1})$$

Preuve

- $\varphi^{(i)}(s_{i+1}) = \varphi^{(i)}(s_i) + |V_i|$
- $\varphi^{(i-1)}(s_{i+1}) = \varphi^{(i-1)}(s_i) + |B_i^1| = \varphi^{(i)}(s_i) + |B_i^1|$

Donc :

$$\varphi^{(i-1)}(s_{i+1}) - \varphi^{(i)}(s_{i+1}) = |B_i^1| - |V_i|$$

De plus, $L_{i+1} = \{j \in \{2, \cdots, k_i\}, \varphi^{(i-1)}(h_{i+1}^1) < \varphi^{(i-1)}(h_i^j)\} = \{2, \cdots, k_i\}$.

Donc, $\sum_{j \in L_{i+1}} |B_i^j| = |V_i| - |B_i^1|$, et on obtient le résultat voulu en rappelant le lemme 11.

□

Théorème 3.2.4 (Dominance des numérotations par blocs). *Soit* $G \in$ *r-2TSPG le résultat de la composition de* K *graphes* $G_i \in$ *r-2TSPG*, $\forall i \in \{1, \cdots, K\}$, *et soit* φ *une numérotation de* G. *Il est possible de construire une numérotation par blocs de coût moindre :*

$$\mathrm{DSC}(\varphi', G) \leq \mathrm{DSC}(\varphi, G)$$

Preuve Dans ce qui précède nous avons présenté une suite de transformations qui, à partir d'une numérotation $\varphi = \varphi^{(0)}$, conduit à une numérotation par blocs $\varphi^{(K-1)}$. Les lemmes 7, 8, 10, 12 montrent qu'à chaque étape $i \in \{1, \cdots, K-1\}$, on a :

$$\begin{aligned}
\mathrm{DSC}(\varphi^{(i)}, G) - \mathrm{DSC}(\varphi^{(i-1)}, G) &= \mathrm{DSC}(\varphi^{(i)}, s) - \mathrm{DSC}(\varphi^{(i-1)}, s) \\
&+ \mathrm{DSC}(\varphi^{(i)}, G_i) - \mathrm{DSC}(\varphi^{(i-1)}, G_i) \\
&+ \mathrm{DSC}(\varphi^{(i)}, G_{i+1}) - \mathrm{DSC}(\varphi^{(i-1)}, G_{i+1}) \\
&+ \sum_{l=i+2}^{k} \mathrm{DSC}(\varphi^{(i)}, G_l) - \mathrm{DSC}(\varphi^{(i-1)}, G_l) \\
&\leq 0
\end{aligned}$$

Par conséquent, $\mathrm{DSC}(\varphi^{(K-1)}, G) \leq \mathrm{DSC}(\varphi^{(0)}, G)$, ce qui nous amène au résultat voulu. □

Numérotation par blocs optimale

Dans ce qui suit, nous montrons comment calculer une numérotation optimale en temps polynomial. Dans un premier temps, nous montrons que dans le cas des compositions parallèles binaires, le coût ne dépend pas de l'ordre de numérotation des blocs. Ensuite, nous montrons comment le problème de la composition n-aire peut être ramené à un problème d'ordonnancement polynomial connu.

Composition binaire On considère maintenant que G est le résultat de la composition parallèle de $G_1 = (V_1, A_1)$ et $G_2 = (V_2, A_2)$.

Théorème 3.2.5. *Soit φ_1 la numérotation par blocs telle que V_1 est numéroté avant V_2, et soit φ_2 l'autre numérotation (voir figure 3.19). Si les deux numérotations vérifient :*

$$\forall (x, y) \in V_1 \times V_1, \varphi_1(x) < \varphi_1(y) \Leftrightarrow \varphi_2(x) < \varphi_2(y)$$

$$\forall (x, y) \in V_2 \times V_2, \varphi_1(x) < \varphi_1(y) \Leftrightarrow \varphi_2(x) < \varphi_2(y)$$

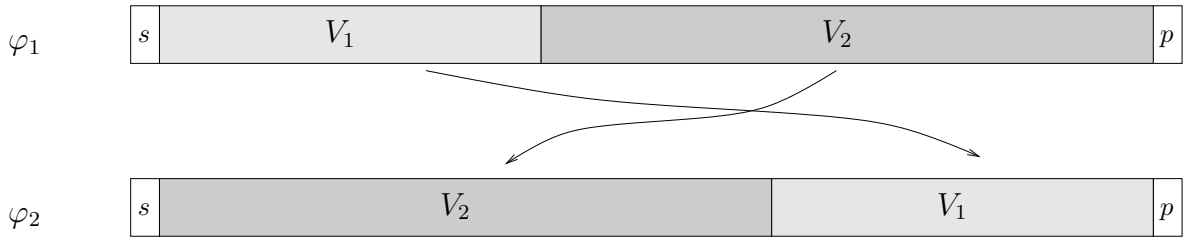


FIG. 3.19 – Numérotations pour le théorème 3.2.5

Alors :

$$\mathrm{DSC}(\varphi_1, G) = \mathrm{DSC}(\varphi_2, G)$$

.

Preuve $\forall u \in \{V_1 \cup V_2\} - \{p_1, p_2\}$, $\mathrm{DSC}(\varphi_1, u) = \mathrm{DSC}(\varphi_2, u)$, ainsi :

$$\mathrm{DSC}(\varphi_1, G_1) - \mathrm{DSC}(\varphi_2, G_1) = \mathrm{DSC}(\varphi_1, p_1) - \mathrm{DSC}(\varphi_2, p_1) = |V_2|, \text{ et}$$

$$\mathrm{DSC}(\varphi_1, G_2) - \mathrm{DSC}(\varphi_2, G_2) = \mathrm{DSC}(\varphi_1, p_2) - \mathrm{DSC}(\varphi_2, p_2) = -|V_1|$$

De plus, $\mathrm{DSC}(\varphi_1, s) = |V_1| + 1$ et $\mathrm{DSC}(\varphi_2, s) = |V_2| + 1$. Donc,

$$\begin{aligned} \mathrm{DSC}(\varphi_1, G) - \mathrm{DSC}(\varphi_2, G) &= \mathrm{DSC}(\varphi_1, s) - \mathrm{DSC}(\varphi_2, s) \\ &+ \mathrm{DSC}(\varphi_1, G_1) - \mathrm{DSC}(\varphi_2, G_1) \\ &+ \mathrm{DSC}(\varphi_1, G_2) - \mathrm{DSC}(\varphi_2, G_2) \\ &= |V_1| + 1 - |V_2| - 1 + |V_2| - |V_1| = 0 \end{aligned}$$

□

Théorème 3.2.6. *Une numérotation optimale peut être trouvée en temps constant (au niveau des blocs) si G est construit à l'aide de composition binaires.*

Preuve Le théorème 3.2.4 prouve la dominance des numérotations par blocs. De plus, le théorème 3.2.5 prouve que l'ordre de parcours des blocs n'a pas d'importance. On conclut que l'algorithme nécessite $\mathcal{O}(1)$ opérations. □

Composition n-aire On considère maintenant $G = (V, A)$, la composition parallèle de K graphes $G_i = (V_i, A_i), i \in \{1, \cdots, K\}$. Par le théorème de dominance (3.2.4) on restreint l'ensemble des numérotations possibles de G aux numérotations par blocs telles que $\forall (u, v) \in V_i \times V_i, \varphi_i(u) < \varphi_i(v) \Rightarrow \varphi(u) < \varphi(v)$, où φ_i est une numérotation optimale de $G_i, i \in \{1, \cdots, K\}$. Par conséquent, on considèrera à partir de maintenant (pour des raisons de clarté) seulement des **fonctions d'étiquetage de blocs**. Une fonction d'étiquetage de blocs (c'est à dire ici de sous-graphes G_i) $\mathcal{N}_\varphi$ est définie pour une numérotation φ de telle manière que pour tout $i \in \{1, \cdots, K\}$, $\mathcal{N}_\varphi(i)$ est l'indice du sous-graphe placé en i-ème position dans φ. La fonction réciproque $\mathcal{N}_\varphi^{-1}(i)$ représente alors la place de G_i dans φ. Un exemple est illustré en figure 3.20. Pour plus de clarté, et quand il n'y aura pas d'ambiguïté, nous écrirons $\mathcal{N}$ à la place de $\mathcal{N}_\varphi$, et nous appellerons $\mathcal{N}$ numérotation par un léger abus de langage.

$$\forall (i, j) \in \{1, \cdots, K\}^2, \mathcal{N}^{-1}(i) < \mathcal{N}^{-1}(j) \Leftrightarrow \forall (u, v) \in V_i \times V_j, \varphi(u) < \varphi(v).$$

La notation $\mathrm{DSC}(\mathcal{N}, G)$ (*resp.* $\mathrm{DSC}(\mathcal{N}, u)$) représente le coût de la numérotation $\mathcal{N}$ pour G (*resp.* $u \in V$).

(a) Numérotation

i	1	2	3	4
$\mathcal{N}_\varphi(i)$	2	3	1	4

(b) Fonction d'étiquetage de blocs correspondante

FIG. 3.20 – Exemple de fonction d'étiquetage de blocs

$$\begin{aligned}\mathrm{DSC}(\mathcal{N},G) &= \mathrm{DSC}(\mathcal{N},s)+\sum_{i=1}^{K}\mathrm{DSC}(\mathcal{N},G_i)\\ &= \mathrm{DSC}(\mathcal{N},s)+\sum_{i=1}^{K}[\sum_{u\in V_i-\{p_i\}}\mathrm{DSC}(\mathcal{N},u)+\mathrm{DSC}(\mathcal{N},p_i)]\end{aligned}$$

Lemme 13. $\sum_{i=1}^{K}\sum_{u\in V_i-\{p_i\}}\mathrm{DSC}(\mathcal{N},u)$ *ne dépend pas de* $\mathcal{N}$.

Preuve Puisque $G\in$ r-2TSPG, pour tout arc $(u,v), u\in V_i-\{p_i\}, v\in V$ on a en fait $v\in V_i$. □

L'objectif devient maintenant de trouver $\mathcal{N}^\star$ vérifiant :

$$\mathrm{DSC}'(\mathcal{N}^\star,G)=\min_{\mathcal{N}}(\mathrm{DSC}(\mathcal{N},s)+\sum_{i=1}^{K}\mathrm{DSC}(\mathcal{N},p_i))$$

Nous prouvons dans la suite que ce problème peut se ramener à un problème d'ordonnancement connu.

Lemme 14. $\forall i, \mathrm{DSC}(\mathcal{N},p_i)=\varphi(p)-\varphi(p_i)=(\sum_{j>i}|V_{\mathcal{N}(j)}|)+1$.

Preuve Pour tout i, p_i n'a qu'un successeur (p). De plus, les sommets numérotés entre p_i et p sont exactement ceux de$\bigcup_{j>i}V_{\mathcal{N}(j)}$. □

Lemme 15. $\mathrm{DSC}(\mathcal{N},s)=\sum_{j=1}^{K}|V_j|-|V_{\mathcal{N}(K)}|+1$.

Preuve Le dernier successeur de s est le sommet source de $G_{\mathcal{N}(K)}$. □

Donc, par les lemmes 14 et 15 :

$$\begin{aligned}
\mathrm{DSC}(\mathcal{N},s)+\sum_{i=1}^{K}\mathrm{DSC}(\mathcal{N},p_i) &= \sum_{j=1}^{K}|V_j|-|V_{\mathcal{N}(K)}|+1+\sum_{i=1}^{K-1}(\sum_{j=i+1}^{K}|V_{\mathcal{N}(j)}|+1)\\
&= \sum_{j=1}^{K}|V_j|+\sum_{i=1}^{K-2}\sum_{j=i+1}^{K}|V_{\mathcal{N}(j)}|+\sum_{i=1}^{K-1}1+1\\
&= \sum_{j=1}^{K}|V_j|+\sum_{i=1}^{K-2}\sum_{j=i+1}^{K-1}|V_{\mathcal{N}(j)}|+\sum_{i=1}^{K-2}|V_{\mathcal{N}(K)}|\\
&+ (\sum_{i=1}^{K-1}1)+1
\end{aligned}$$

- Le premier terme ne dépend pas de $\mathcal{N}$,
- Le second terme sera exprimé dans la suite comme le critère d'un problème d'ordonnancement,
- Le troisième terme dépend du dernier sous-graphe ($V_{\mathcal{N}(K)}$),
- Les derniers termes sont constants.

Pour pouvoir minimiser $\sum_{i=1}^{K-2}\sum_{j=i+1}^{K}|V_{\mathcal{N}(j)}|$, nous considérons le problème d'ordonnancement suivant :

Problème: $1||\sum C_j$

Instance: Soient $T_j, 1\leq j\leq Q$, Q tâches de durée L_j, et 1 processeur.

Question: Si C_j est la date de fin de la tâche $T_j, j\in\{1,\cdots,Q\}$, construire un ordonnancement minimisant $\sum_{j=1}^{Q}C_j$.

Le théorème 3.2.7 montre que ce problème est simple :

Théorème 3.2.7. *$\sum_{j=1}^{Q}C_j$ peut être minimisée en temps polynomial [Smi56], [Bak74].*

Soit $\mathcal{I}$ une instance de minDSC associée à la composition parallèle de K sous-graphes $G_i, i\in\{1,\cdots,K\}$, dont les numérotations optimales $\varphi_i, i\in\{1,\cdots,K\}$ sont données. nous nous intéressons aux numérotations telles que $\mathcal{N}(K)=i_0, i_0\in\{1,\cdots,K\}$. L'instance correspondante du problème d'ordonnancement est donnée par $Q=K-1$ tâches de durées $L_j=|V_j|, j\in\{1,\cdots,K\}-\{i_0\}$.

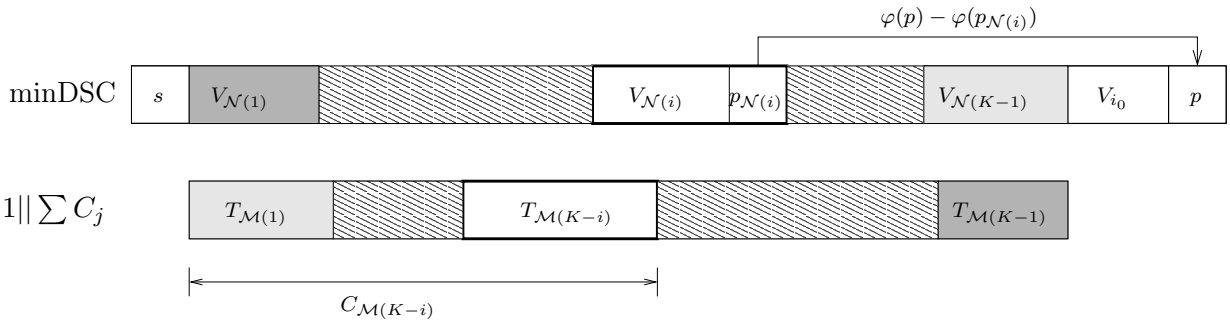


FIG. 3.21 – Réduction de r-2TSPG-minDSC à $1||\sum C_j$

A $\mathcal{N} : \{1, \cdots, K\} \rightarrow \{1, \cdots, K\}$, nous associons une numérotation $\mathcal{M} : \{1, \cdots, K-1\} \rightarrow \{1, \cdots, K\} - \{i_0\}$ au problème d'ordonnancement définie comme suit :

$$\mathcal{M}(i) = \mathcal{N}(K-i), \forall i \in \{1, \cdots, K-1\}$$

La fonction d'étiquetage de tâches $\mathcal{M}$ correspond à l'inverse de la numérotation de graphe $\mathcal{N}$ (voir figure 3.21). Pour un étiquetage de tâches donné $\mathcal{M}$, les valeurs C_j représentent les dates de fin des tâches correspondantes.

Lemme 16. $\displaystyle\sum_{j=1}^{K-1} \sum_{l=j+1}^{K-1} L_{\mathcal{N}(j)} = \sum_{j=1}^{K-1} C_j - \sum_{j=1}^{K-1} L_{\mathcal{N}(j)}$

Preuve Tout d'abord, nous déterminons les valeurs des C_j.

$$\begin{aligned} C_{\mathcal{M}(j)} = \sum_{i=1}^{j} L_{\mathcal{M}(i)} \quad &\Leftrightarrow \quad C_j = \sum_{i=1}^{\mathcal{M}^{-1}(j)} L_{\mathcal{M}(i)} \\ &\Leftrightarrow \quad C_j = \sum_{i=1}^{\mathcal{M}^{-1}(j)} L_{\mathcal{N}(K-i)} \\ &\quad\ \text{Soit } l = K - i, \text{ donc puisque } K - \mathcal{M}^{-1}(j) = \mathcal{N}^{-1}(j) \\ &\Leftrightarrow \quad C_j = \sum_{l=\mathcal{N}^{-1}(j)}^{K-1} L_{\mathcal{N}(l)} \end{aligned}$$

$$\begin{aligned}
\sum_{j=1}^{K-1} C_j &= \sum_{j=1}^{K-1} \sum_{l=\mathcal{N}^{-1}(j)}^{K-1} L_{\mathcal{N}(l)} \\
&= \sum_{j=1}^{K-1} \sum_{l=j}^{K-1} L_{\mathcal{N}(l)} \text{ car } \mathcal{N}^{-1}(j) \text{ parcourt } 1 \cdots K-1 \\
&= \sum_{j=1}^{K-1} \left(\sum_{l=j+1}^{K-1} L_{\mathcal{N}(l)} + L_{\mathcal{N}(j)} \right) \\
\Leftrightarrow \sum_{j=1}^{K-1} \sum_{l=j+1}^{K-1} L_{\mathcal{N}(l)} &= \sum_{j=1}^{K-1} \left(C_j - L_{\mathcal{N}(j)} \right) = \sum_{j=1}^{K-1} C_j - \sum_{j=1}^{K-1} L_{\mathcal{N}(j)}
\end{aligned}$$

□

Lemme 17. *La minimisation de* $\sum_{j=1}^{K-1} \sum_{l=j+1}^{K-1} L_{\mathcal{N}(l)}$ *telle que* $\mathcal{N}(K) = i_0$ *peut être réalisée en temps polynomial.*

Preuve $\sum_{j=1}^{K-1} L_{\mathcal{N}(j)}$ est constante, donc le théorème 3.2.7 nous permet de conclure. □

Théorème 3.2.8. *Le score minimal pour* DSC *peut être atteint en* $\mathcal{O}(K^2 \times \log(K))$ *operations, où* K *est le nombre maximal de graphes composés en parallèle pour la construction de* G*.*

Preuve Minimiser $\sum_{i=1}^{K-1} \sum_{j>i}^{K-1} L_{\mathcal{N}(j)} + \sum_{i=1}^{K-1} L_{\mathcal{N}(K)}$ conduit à un DSC(φ, G) optimal. Comme on l'a vu dans le lemme 16, $\sum_{i=1}^{K-1} L_{\mathcal{N}(K)}$ peut prendre K valeurs différentes, selon le dernier bloc $i_0/\mathcal{N}(K) = i_0$. De plus, pour chacune de ces cas une numérotation optimale $\mathcal{N}_{i_0}^{\star}$ for $\sum_{i=1}^{K-1} \sum_{j>i}^{K-1} L_{\mathcal{N}(j)}$ peut être atteinte en temps polynomial (théorème 3.2.7). L'algorithme pour notre problèmes consiste alors à rechercher la meilleure parmi K numérotation. Par conséquent, le nombre d'opérations réalisées est $\mathcal{O}(K^2 \times \log(K))$. □

Extension du résultat

Afin de tenter de relâcher les hypothèses, nous avons essayé d'étudier la dominance des numérotations par blocs dans les classes 2TSP et SP. Ces travaux ont mené à deux contre-exemples, comme montré en figures 3.22 et 3.23.

3.2.4 Ordres intervalles

Les ordres intervalles consituent une classe très étudiée en théorie des graphes [Gol04], car ils sont reliés aux graphes parfaits. Dans le cadre de nos travaux, l'intérêt de cette

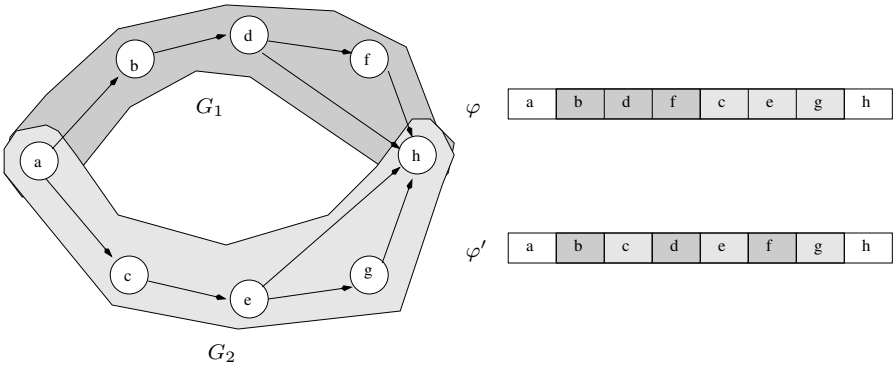


FIG. 3.22 – Contre exemple pour les graphes 2-terminaux $\mathrm{DSC}(\varphi) = 18 > \mathrm{DSC}(\varphi') = 17$

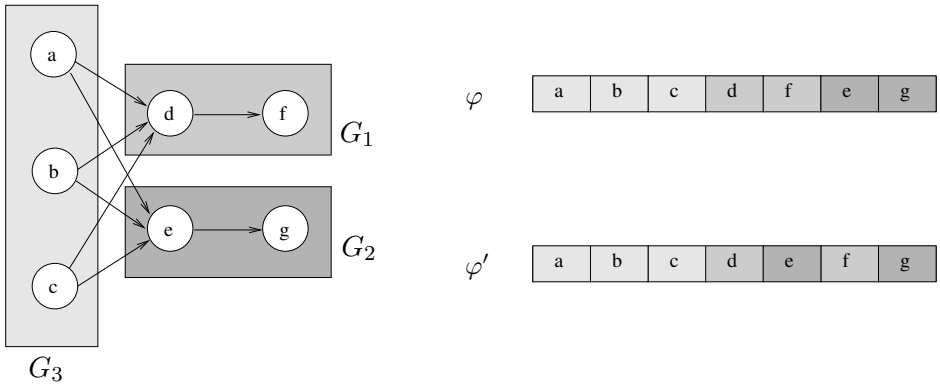


FIG. 3.23 – Contre exemple pour les graphes SP $\mathrm{DSC}(\varphi) = 14 > \mathrm{DSC}(\varphi') = 13$

structure (pour la version orientée) est que les durées de vie des variables lors de l'exécution d'un programme peuvent être vues comme une famille partiellement ordonnée de la droite réelle. Dans cette partie, nous montrons que le problème minDSC peut être résolu en temps polynomial si G est un ordre intervalles.

Définition et propriétés des ordres intervalles

Définition 8 (ordre intervalles). *Soit $\mathcal{I} = (I_k)_{k=1\cdots n}$ une famille d'intervalles de $\mathbb{R}$ de la forme $I_k = [début_k, fin_k]$. L'***ordre intervalles** $G = (V, A)$ *associé à $\mathcal{I}$ est un graphe orienté construit de la façon suivante :*

- *A chaque intervalle I_k de $\mathcal{I}$, $k \in \{1, \cdots, n\}$, on associe de façon bijective un sommet v_k de G,*
- *A chaque couple d'intervalles $(I_j, I_k) \in \mathcal{I} \times \mathcal{I}$, on associe de façon bijective un arc (v_j, v_k) si $fin_j < début_k$.*

Un exemple d'ordre intervalles est montré en figure 3.24.

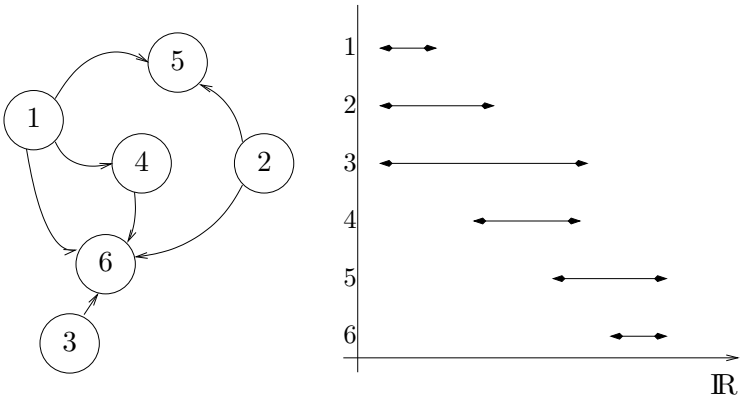


FIG. 3.24 – Exemple d'ordre intervalles

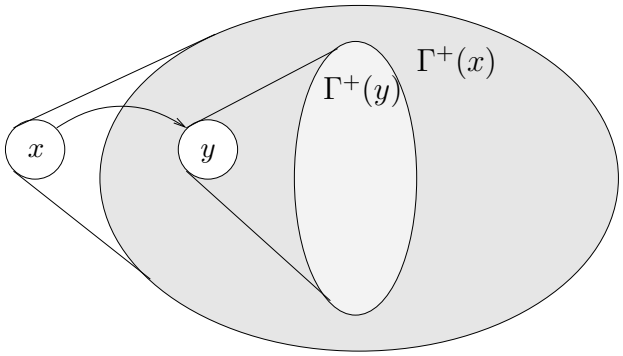


FIG. 3.25 – Exemple pour le corollaire 3

Soit $G = (V, A)$ un ordre intervalles. Pour tout sommet $v \in V$, on note $\Gamma^+(v) = \{u \in V, (v, u) \in A\}$, et $\delta^+(v) = |\Gamma^+(v)|$ son degré sortant. Soit φ une numérotation des sommets de G. On note finalement DSC(φ) le coût de φ au sens de DSC.

Propriété 3 (ordre intervalles [PY79]). *Si G est un* **ordre intervalles**, *alors pour tout couple $(x, y) \in V \times V$, on a soit $\Gamma^+(x) \subseteq \Gamma^+(y)$, soit $\Gamma^+(y) \subseteq \Gamma^+(x)$.*

Corollaire 3. *Si $(x, y) \in A$, alors $\Gamma^+(y) \subset \Gamma^+(x)$.*

Preuve L'inclusion opposée est impossible car $y \notin \Gamma^+(y)$ (voir figure 3.25). □

Définition 9. *On appelle* **sommet puits** *tout sommet $x \in V$ tel que $\delta^+(x) = 0$. On note S l'ensemble des sommets puits de G.*

Les sommets puits du graphe de la figure 3.24 sont mis en valeur en figure 3.26.

Lemme 18. *Soit $x \in V - S$. Alors il existe un sommet puits s tel que $(x, s) \in A$.*

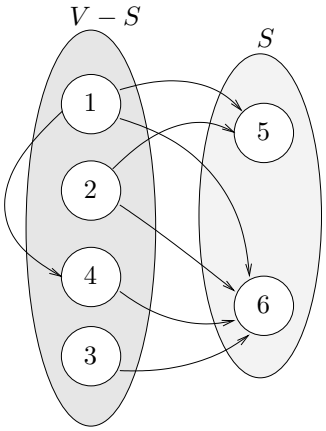


FIG. 3.26 – Sommets puits du graphe de la figure 3.24

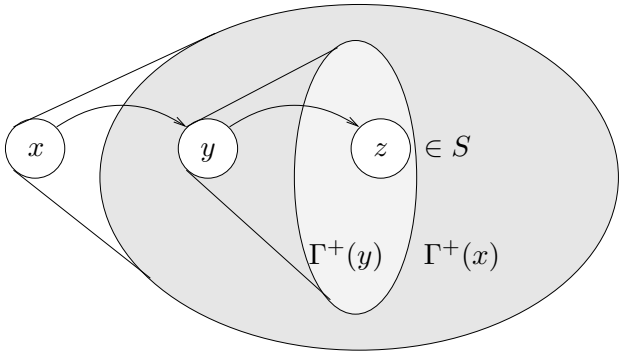


FIG. 3.27 – Exemple pour le lemme 19

Preuve Par contradiction, soit $x \in V - S$ le dernier sommet (*i.e.* maximal au sens de φ) ne vérifiant pas le lemme. Soit $y \in \Gamma^+(x)$. Si $y \in S$, le lemme est prouvé. Sinon, on sait par maximalité de $\varphi(x)$ qu'il existe $z \in S$ tel que $(y, z) \in A$. Comme G est un ordre intervalles, on sait que $\Gamma^+(y) \subseteq \Gamma^+(x)$. Par conséquent, on a que $z \in \Gamma^+(x)$. □

Lemme 19. *Soit $x \in V - S$, et soit $y \in \Gamma^+(x)$ tel que $\varphi(y) = \max_{u \in \Gamma^+(x)} \varphi(u)$ (y est le dernier successeur de x au sens de φ). Alors $y \in S$.*

Preuve Soit $x \in V - S$ le dernier sommet (*i.e.* maximal au sens de φ) ne vérifiant pas le lemme. $y \notin S$ par hypothèse sur x. Soit alors z le dernier successeur de y (voir figure 3.27). Par maximalité de $\varphi(x)$, on sait que $z \in S$.

- Si $z \in \Gamma^+(x) \cap S$, alors $\varphi(z) > \varphi(y)$, ce qui est impossible par construction de y.
- $z \in S - \Gamma^+(x)$ n'est pas possible non plus car alors on a une autre contradiction. En effet, $z \in \Gamma^+(y) \subset \Gamma^+(x)$ (d'après le corollaire 3).

□

Proposition 1. *Pour tout couple $(x, y) \in S$, on a soit $\Gamma^-(x) \subseteq \Gamma^-(y)$, soit $\Gamma^-(y) \subseteq \Gamma^-(x)$.*

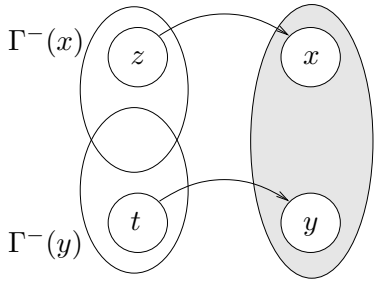


Fig. 3.28 – Exemple pour la preuve de la proposition 1

Preuve Supposons qu'il existe 2 sommets puits $(x, y) \in S$ tels que $\Gamma^-(x) \not\subseteq \Gamma^-(y)$ et $\Gamma^-(y) \not\subseteq \Gamma^-(x)$. Alors $\exists z \in \Gamma^-(x) - \Gamma^-(y)$. On a alors $x \in \Gamma^+(z)$ et $y \notin \Gamma^+(z)$. On a aussi $\exists t \in \Gamma^-(y) - \Gamma^-(x)$. On a alors $y \in \Gamma^+(t)$ et $x \notin \Gamma^+(t)$ (voir figure 3.28). Par conséquent, on a que $x \in \Gamma^+(z) - \Gamma^+(t)$, donc $\Gamma^+(z) \not\subseteq \Gamma^+(t)$. Le même raisonnement sur y nous donne $\Gamma^+(t) \not\subseteq \Gamma^+(z)$. La propriété 3 fondamentale des ordres intervalles est donc contredite. □

Soit $(s_1, s_2) \in S \times S$. On note :

$$s_1 \equiv s_2 \Leftrightarrow \Gamma^-(s_1) = \Gamma^-(s_2)$$

$\equiv$ est une relation d'équivalence. On obtient donc une partition de S d'après les classes d'équivalence pour $\equiv$: $S' \subset S$ est une **classe d'équivalence** pour $\equiv$ si pour tout $(u, v) \in S'^2$, on a $u \equiv v$. On note $(S_i)_{i=1\cdots\sigma}$ les classes, et la partition devient :

$$S = \bigcup_{i=1}^{\sigma} S_i, S_i \cap S_j = \emptyset, \forall i \neq j$$

Pour tout sommet puits s, on note $\{s\}$ sa classe d'équivalence. Par exemple, pour la figure 3.24, on a $S_1 = \{5\}$ et $S_2 = \{6\}$.

Définition 10 (Rang d'une partie de S). *Soit $S' \subset S$, alors on note :*

$$rang(S') := |\Gamma^-(S')|,$$

où $|\Gamma^-(S')| := |\bigcup_{s \in S'} \Gamma^-(s)|$.

Lemme 20. *Soient S_1, S_2 2 classes d'équivalence de S pour la relation $\equiv$. Alors :*

$$(i) rang(S_1) = rang(S_2) \Leftrightarrow (ii) S_1 = S_2$$

Preuve

(ii) $\Longrightarrow$ (i) Trivial

(i) $\Longrightarrow$ (ii) Soit $s_1 \in S_1$ et $s_2 \in S_2$. Alors par la proposition 1, on a soit $\Gamma^-(s_1) \subseteq \Gamma^-(s_2)$, soit $\Gamma^-(s_2) \subseteq \Gamma^-(s_1)$. Comme $\text{rang}(S_1) = \text{rang}(S_2)$, alors on a $|\Gamma^-(S_1)| = |\Gamma^-(S_2)|$, donc $|\Gamma^-(s_1)| = |\Gamma^-(s_2)|$. Par conséquent, s_1 et s_2 sont dans la même classe, donc $S_1 = S_2$.

□

On définit maintenant la relation suivante :

$$S_1 \ll S_2 \Leftrightarrow \text{rang}(S_1) \leq \text{rang}(S_2)$$

Le lemme qui suit montre que l'ensemble $((S_i)_{i\in\{1,\cdots,\delta\}}, \ll)$ est totalement ordonné.

Lemme 21. *La famille $(S_i)_{i=1\cdots\sigma}$ (l'espace quotient de S pour la relation $\equiv$) est totalement ordonnée. De plus, si deux classes S_1 et S_2 vérifient $S_1 \ll \S_2$, alors on a $\Gamma^-(S_1) \subseteq \Gamma^-(S_2)$.*

Preuve On montre d'abord que $\ll$ est une relation d'ordre.
(a) Réflexivité, transitivité : ces propriétés sont triviales,
(b) Antisymétrie : l'antisymétrie est déduite du lemme 20.
Ensuite, soient $s_1 \in S_1$ et $s_2 \in S_2$. Par définition des classes, $|\Gamma^-(s_1)| \leq |\Gamma^-(s_2)|$. On déduit de la proposition 1 que $\Gamma^-(s_1) \subseteq \Gamma^-(s_2)$, et donc $\Gamma^-(S_1) \subseteq \Gamma^-(S_2)$.

□

Corollaire 4. *Il existe un sommet puits s tel que $\Gamma^-(s) = V - S$.*

Preuve Soit S_δ la classe de S de rang maximal, et s un sommet de S_δ. Par le lemme 21, pour toute classe S_i de S , on a $\text{rang}(S_i) \ll \text{rang}(S_\delta)$, et donc $\Gamma^-(S_i) \subseteq \Gamma^-(S_\delta)$. Ainsi, $\Gamma^-(S_\delta) = \Gamma^-(S) = V - S$, et tout sommet $s \in S_\delta$ vérifie donc le corollaire. □

Algorithme polynômial pour la résolution de minDSC

Soit $G = (V, A)$, un ordre intervalles, et soit $l = |S|$, où S désigne l'ensemble des sommets puits de G. On note ceux-ci $s_1 \cdots s_l$, de façon à ce que $\{s_1\} \ll \{s_2\} \ll \cdots \ll \{s_l\}$. On sait par le corollaire 4 que $\Gamma^-(s_l) = V - S$. Par conséquent, tout ordre φ vérifiera que si $\varphi(x) > \varphi(s_l)$ alors $x \in S$. On adoptera alors la notation suivante :

$$\Lambda_\varphi := \{x \in S, \varphi(x) \geq \varphi(s_l)\}$$

On dira que Λ_φ est trié selon les degrés entrants décroissants si $\forall (x, y) \in \Lambda_\varphi \times \Lambda_\varphi, \varphi(x) \leq \varphi(y) \Leftrightarrow \{x\} \gg \{y\}$

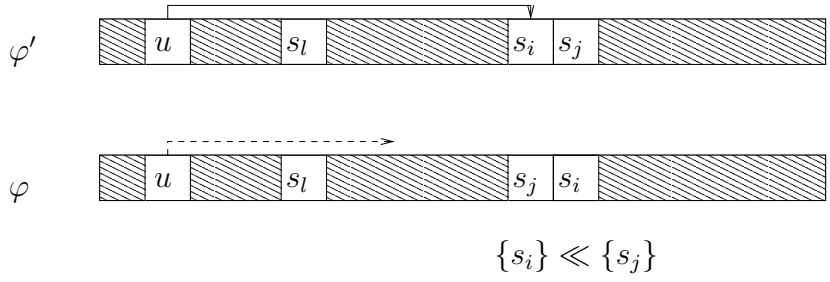


FIG. 3.29 – Transformation de φ' à φ pour le lemme 22

Lemme 22. *Soit φ un ordre. Soit φ' l'ordre obtenu en posant d'une part $\varphi(x) = \varphi'(x) \forall x \notin \Lambda_\varphi$, et d'autre part en triant Λ_φ selon les degrés entrants décroissants. Alors :*

$$\mathrm{DSC}(\varphi') \leq \mathrm{DSC}(\varphi)$$

Preuve Soit φ un ordre. Soit s_i le premier sommet de Λ_φ tel que $\{s_i\} \ll \{s_j\}$, où s_j est le sommet suivant s_i dans φ (*i.e.* $s_j = \varphi^{-1}(\varphi(s_i)+1)$). Soit φ' l'ordre obtenu à partir de φ en inversant s_i et s_j (voir figure 3.29). On se propose de montrer que $\mathrm{DSC}(\varphi') \leq \mathrm{DSC}(\varphi)$.

$$\begin{aligned} \mathrm{DSC}(\varphi, G) - \mathrm{DSC}(\varphi', G) &= \sum_{u \in V-S} \mathrm{DSC}(\varphi, u) - \sum_{u \in V-S} \mathrm{DSC}(\varphi', u) \\ &= \sum_{u \in \Gamma^-(s_j)} \mathrm{DSC}(\varphi, u) - \sum_{u \in \Gamma^-(s_j)} \mathrm{DSC}(\varphi', u) \end{aligned}$$

En effet, pour tout sommet $u \in V - S - \Gamma^-(s_j)$, la position de son dernier successeur n'est pas modifiée, car $\{s_i\} \ll \{s_j\}$ implique par définition et par le lemme 21 que $\Gamma^-(s_i) \subseteq \Gamma^-(s_j)$. Si $u \in \Gamma^-(s_j)$, on a alors 3 cas :

(a) Si le dernier successeur y de u pour φ est tel que $\varphi(y) > \varphi(s_j)$, alors $\mathrm{DSC}(\varphi', u) = \mathrm{DSC}(\varphi, u)$,

(b) Sinon, si $u \in \Gamma^-(s_j) - \Gamma^-(s_i)$, alors s_j est le dernier successeur de u pour φ (sinon le cas précédent aurait été vérifié), et donc $\mathrm{DSC}(\varphi', u) = \mathrm{DSC}(\varphi, u) - 1$,

(c) Enfin, si $u \in \Gamma^-(s_i) \subset \Gamma^-(s_j)$, le dernier successeur de u pour φ est s_j, et le dernier successeur de u pour φ' est s_i. Comme $\varphi(s_j) = \varphi'(s_i)$, on a $\mathrm{DSC}(\varphi', u) = \mathrm{DSC}(\varphi, u)$.

□

Lemme 23. *Les ordres φ tels que :*
(i) Λ_φ est trié selon les degrés entrants décroissants,
(ii) et $\Lambda_\varphi = S$
sont dominants.

Preuve Soit φ'' un ordre. Le lemme 22 nous permet de construire un ordre φ' tel que $\mathrm{DSC}(\varphi') \leq \mathrm{DSC}(\varphi'')$ et que $\Lambda_{\varphi'}$ est trié selon les degrés entrants décroissants. Dans le cas

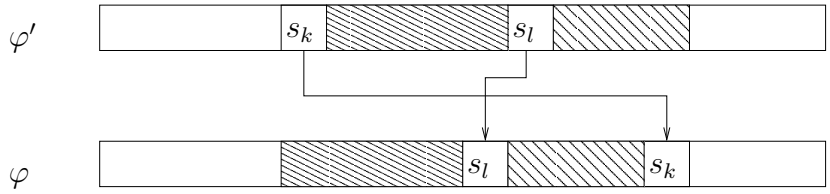


FIG. 3.30 – Transformation de φ' à φ pour le lemme 23

où $\Lambda_\varphi \neq S$, Nous nous proposons de construire un ordre φ tel que $\mathrm{DSC}(\varphi) \leq \mathrm{DSC}(\varphi')$ et $|\Lambda_\varphi| = |\Lambda_{\varphi'}| + 1$.

Soit s_k le sommet de S maximal pour la numérotation φ' et tel que $s_k \notin \Lambda_{\varphi'}$. On construit la numérotation φ à partir de φ' en réalisant l'opération suivante (voir figure 3.30) : on insère s_k dans $\Lambda_{\varphi'}$ à sa place (*i.e.* en respectant l'ordre) pour construire $\Lambda_{\varphi'}$. Les autres éléments ne sont pas modifiés. Les seuls sommets dont le coût est susceptible d'être modifié sont les sommets de $V - S$.

$$\mathrm{DSC}(\varphi) - \mathrm{DSC}(\varphi') = \sum_{u \in V-S} \mathrm{DSC}(\varphi, u) - \sum_{u \in V-S} \mathrm{DSC}(\varphi', u)$$

Soit $u \in V - S$, et soit v' le dernier successeur de u pour φ'. Par le corollaire 4, $u \in \Gamma^-(s_l)$, donc $\varphi(v') \geq \varphi(s_l)$, et $v' \in \Lambda_{\varphi'}$. Soit alors v le dernier successeur de u pour φ. Comme précédemment, $v \in \Lambda_\varphi$.

Si $\delta^-(v') < \delta^-(s_k)$, alors d'une part par le lemme 21, et d'autre part par la définition des classes d'équivalence, on a $u \in \Gamma^-(v') \subseteq \Gamma^-(s_k)$. Donc $\varphi(s_k) < \varphi(v')$, et $v = v'$. Comme $\varphi(u) < \varphi(s_k) < \varphi(v)$ et $\varphi'(u) < \varphi'(s_k) < \varphi'(v)$, le coût de u n'est donc pas modifié.

Sinon, $\delta^-(v') \geq \delta^-(s_k) \implies \varphi(v') \leq \varphi(s_k)$, et :

(i) Si $v = v'$, alors $\varphi'(v) \geq \varphi(v)$, et donc $\mathrm{DSC}(\varphi', u) - \mathrm{DSC}(\varphi, u) \geq 0$,
(ii) Si $v \neq v'$, alors $v = s_k$ car $\delta^-(v') \geq \delta^-(s_k)$. Soit alors s_r le sommet de Λ_φ tel que $\varphi(s_r) + 1 = \varphi(s_k)$ (*i.e.* numéroté juste avant dans φ) (voir figure 3.31). Alors $\Gamma^-(s_k) \subset \Gamma^-(s_r)$, et donc $u \in \Gamma^-(s_r)$.
On en déduit que $v' = s_r$. Comme de plus $\varphi(s_k) = \varphi'(s_r)$, $\mathrm{DSC}(\varphi', u) = \mathrm{DSC}(\varphi, u)$.

□

Lemme 24. *Soit φ un ordre tel que $\Lambda_\varphi = S$. Soit φ' un ordre vérifiant $\forall x \in S, \varphi'(x) = \varphi(x)$, ainsi que $\Lambda_{\varphi'} = \Lambda_\varphi = S$. Alors* $\mathrm{DSC}(\varphi) = \mathrm{DSC}(\varphi')$.

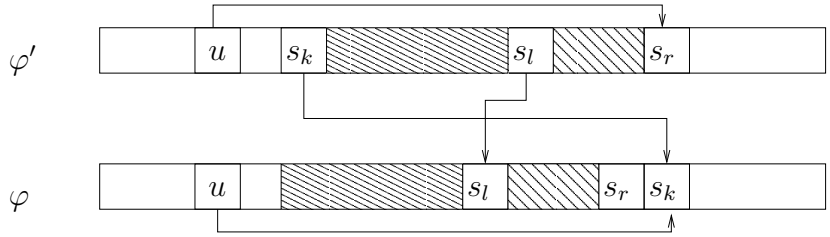


FIG. 3.31 – Modification du coût de u pour le lemme 23

Preuve Pour tout $v \in V - S$, d'après le lemme 19, son dernier successeur est dans S. De plus, par définition de φ et φ', ce dernier successeur est numéroté à la même place par φ et φ'. On le note s_v. On a alors :

$$\begin{aligned} \mathrm{DSC}(\varphi) &= \sum_{v \in V-S} (\varphi(s_v) - \varphi(v)), \\ \text{et } \mathrm{DSC}(\varphi') &= \sum_{v \in V-S} (\varphi'(s_v) - \varphi'(v)) \\ \text{Par conséquent, } \mathrm{DSC}(\varphi) - \mathrm{DSC}(\varphi') &= \sum_{v \in V-S} \varphi'(v) - \sum_{v \in V-S} \varphi(v) \end{aligned}$$

Comme $\Lambda_\varphi = \Lambda_{\varphi'} = S$, et comme φ et φ' sont des bijections, les termes des deux sommes sont les valeurs de $\{1, \cdots, |V - S|\}$, à l'ordre près. Donc par commutativité de l'addition on a $\sum_{v \in V-S} \varphi'(v) = \sum_{v \in V-S} \varphi(v)$, et $\mathrm{DSC}(\varphi) = \mathrm{DSC}(\varphi')$. □

Théorème 3.2.9. *Soit $G = (V, A)$ un ordre intervalles orienté. Une numérotation optimale au sens du score DSC peut être atteinte en un nombre polynomial d'opérations.*

Preuve L'algorithme consiste simplement à trier les sommets de S selon leur degré entrant décroissant (lemme 23). Les autres sommets peuvent être numérotés selon un ordre topologique quelconque (lemme 24). □
Dans le cas du graphe de la figure 3.24, les sommets de S sont 5 et 6, de degrés entrants respectifs 2 et 4. On peut donc calculer une numérotation minimisant le score DSC comme illustré en figure 3.32 (seuls les arcs vers le dernier successeur de chaque sommet sont représentés). Le score obtenu est de 13.

3.3 Conclusion

Nous avons montré que la minimisation des deux scores UCS et DSC est un problème difficile dès les graphes de profondeur 2 (3.1). Nous avons cependant montré quatre classes

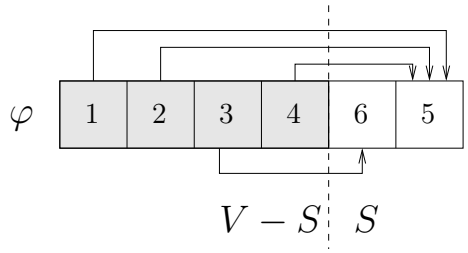


Fig. 3.32 – Une numérotation optimale du graphe de la figure 3.24

de graphes pour lesquelles le problème peut être résolu par des algorithmes nécessitant un nombre polynomial d'opérations (3.2). Dans le chapitre suivant, nous allons montrer comment l'étude de ces cas théoriques nous a permis d'obtenir des heuristiques performantes dans des cas réels.

Les principaux axes de recherche en cours concernent la complexité du problème pour les classes de graphes série-parallèles pour lesquelles les propriétés de composition ne sont pas exploitables telles quelles. Il existe aussi d'autres classes susceptibles de mener à des résultats intéressants. C'est le cas de la classe des ordres quasi-intervalles [Mou03], qui contient à la fois les ordres intervalles et une certaine classe de graphes série-parallèles.

D'autre part, nous voulons aussi étendre les résultats de polynomialité qui ne l'ont pas encore été au problème minUCS.

Le dernier point que nous nous proposons d'explorer concerne la description hiérarchique des systèmes. Nous voulons chercher des méthodes hybrides, permettant de numéroter des graphes décrits hiérachiquement avec des méthodes variant selon le niveau d'abstraction de leur décomposition.

Chapitre 4

Résultats expérimentaux

Dans ce chapitre, nous présentons les résultats d'expériences. Tout d'abord (section 4.1), nous montrons les heuristiques qui ont été élaborées et implémentées. Dans un second temps (section 4.2), nous présentons le protocole opératoire qui a été implémenté pour mettre en œuvre différentes expériences. Finalement, nous montrons en section 4.3 la comparaison du score DSC d'un graphe avec le temps d'exécution du programme correspondant. Cette série d'expériences est destinée à montrer la pertinence de nos modèles par rapport à une exécution dans un environnement réel. Finalement, nous montrons le comportement des heuristiques que nous avons élaborées par rapport à celle qui est actuellement utilisée (une numérotation aléatoire). Les résultats sont positifs, puisqu'on constate à la fin que des baisses de temps d'exécution de 20% peuvent être atteintes.

4.1 Heuristiques

Cette section est destinée à montrer au travers d'un exemple les différentes stratégies permettant de numéroter un graphe orienté sans circuit. Après avoir présenté les méthodes les plus simples, nous montrons deux méthodes spécialement élaborées pour minimiser les critères DSC et UCS. En guise d'exemple, une numérotation du graphe de la figure 4.1 est calculée pour chaque heuristique.

4.1.1 Heuristique Random

Les heuristiques actuellement utilisées dans des simulateurs comme CASS [Hom01] reposent sur un ordre topologique aléatoire du graphe de précédence. A chaque étape i, l'algorithme, dont une implémentation possible est présentée dans le tableau 4.1, numérote un sommet choisi aléatoirement parmi l'ensemble des candidats $L(i)$, ensemble composé

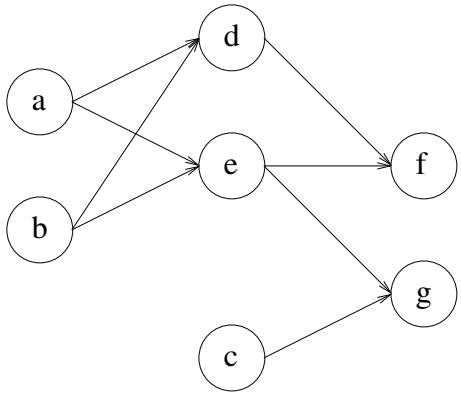


Fig. 4.1 – Exemple de graphe pour les différents algorithmes de numérotation

Algorithme orderRANDOM **:**
Entrée : Un graphe G
Sortie : φ une numérotation de G
$i \leftarrow 1$
$L(i) \leftarrow \{$les sommets sans prédécesseurs de $G\}$
Tant que $L(i) \neq \emptyset$
  Choisir u aléatoirement dans $L(i)$
  $\varphi(u) \leftarrow i$
  $P \leftarrow$les successeurs de u dont tous les prédécesseurs
    ont déjà été numérotés
  $(L(i+1) \leftarrow L(i) \backslash \{u\}) \cup P$
  $i \leftarrow i + 1$
fintantque
finalgo

Tab. 4.1 – Heuristique RANDOM

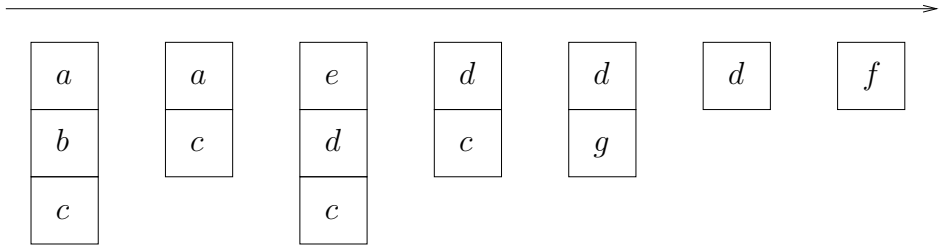
des sommets dont tous les prédécesseurs ont déjà été numérotés.

4.1.2 Parcours en largeur et en profondeur

Les stratégies les plus simples pour améliorer cette méthode selon le critère à optimiser consistent, lors de chaque étape, à orienter le choix du sommet à numéroter. Si on note S l'ensemble des *sommets sources*

$$S := \{u \in V, \Gamma^-(u) = \emptyset\},$$

pour tout sommet $u \in V$, le *niveau* de V est défini comme le nombre d'arcs d'un plus long chemin de S à u. Une stratégie de parcours en profondeur, qui est équivalent dans



$\varphi(b) = 1 \;\; \varphi(a) = 2 \;\; \varphi(e) = 3 \;\; \varphi(c) = 4 \;\; \varphi(g) = 5 \;\; \varphi(d) = 6 \;\; \varphi(f) = 7$

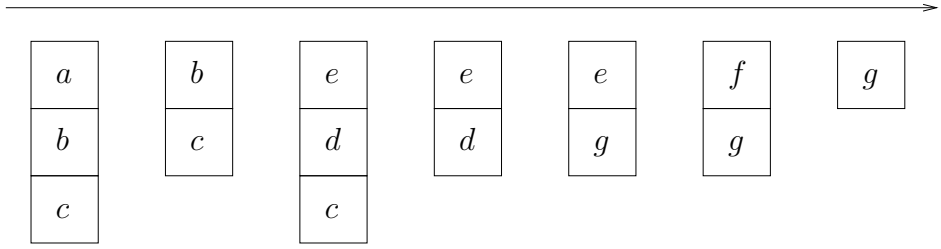
FIG. 4.2 – Evolution des candidats pour la numérotation Random du graphe de la figure 4.1

Algorithme orderFIFO **:**
Entrée : Un graphe G
Sortie : φ une numérotation de G
$i \leftarrow 1$
$L(i) \leftarrow \{$les sommets sans prédécesseurs de $G\}$
Tant que $L(i) \neq \emptyset$
    Choisir u aléatoirement parmi les sommets
    de plus faible niveau dans $L(i)$
    $\varphi(u) \leftarrow i$
    $P \leftarrow$les successeurs de u dont tous les prédécesseurs
        ont déjà été numérotés
    $(L(i+1) \leftarrow L(i) \backslash \{u\}) \cup P$
    $i \leftarrow i + 1$
fintantque
finalgo

TAB. 4.2 – Heuristique FIFO

notre cas à un algorithme LIFO (*resp.* en largeur, ou FIFO) consiste à chaque étape à choisir parmi les candidats un sommet de niveau maximum (*resp.* minimum).

Nous avons testé ces stratégies contre une numérotation aléatoire, sans améliorer de façon significative les scores (voir figure 4.8). Ce constat nous a mené à développer plusieurs heuristiques destinées à prendre encore mieux en compte la structuture de nos modèles, leur point commun étant que, dans tous les cas, un sommet ne doit pas être numéroté trop longtemps après la numérotation de ses prédécesseurs.



$\varphi(a) = 1 \;\; \varphi(b) = 2 \;\; \varphi(c) = 3 \;\; \varphi(d) = 4 \;\; \varphi(e) = 5 \;\; \varphi(f) = 6 \;\; \varphi(g) = 7$

FIG. 4.3 – Evolution des candidats pour la numérotation FIFO du graphe de la figure 4.1

Algorithme orderLIFO **:**
Entrée : Un graphe G
Sortie : φ une numérotation de G
$i \leftarrow 1$
$L(i) \leftarrow \{$les sommets sans prédécesseurs de $G\}$
Tant que $L(i) \neq \emptyset$
    Choisir u aléatoirement parmi les sommets
    de plus haut niveau dans $L(i)$
    $\varphi(u) \leftarrow i$
    $P \leftarrow$les successeurs de u dont tous les prédécesseurs
        ont déjà été numérotés
    $(L(i+1) \leftarrow L(i) \backslash \{u\}) \cup P$
    $i \leftarrow i + 1$
fintantque
finalgo

TAB. 4.3 – Heuristique LIFO

4.1.3 Heuristique HNU

Pour cette nouvelle heuristique, appelée *Heuristique de numérotation urgente (HNU)*, l'ensemble $L(i)$ des candidats à l'étape $i \in \{1, \cdots, n\}$ est composé de sommets dont la numérotation est urgente. Cette notion est bien évidemment très subjective, mais l'idée sous-jacente est qu'à chaque étape on choisit parmi les candidats un sommet qu'on veut numéroter. Quand ce n'est pas possible, tous ses prédécesseurs non encore numérotés deviennent des candidats, et leur numérotation devient urgente. La différence avec les heuristiques présentées ci-dessus réside en effet dans le fait que les candidats ne sont pas forcément des sommets dont tous les prédécesseurs ont déjà été numérotés. Un exemple d'exécution de cette heuristique est montré en figure 4.5, et l'algorithme détaillé est présenté au tableau 4.4.

L'ensemble des candidats est initialisé avec un sommet sans prédécesseur. L'algorithme

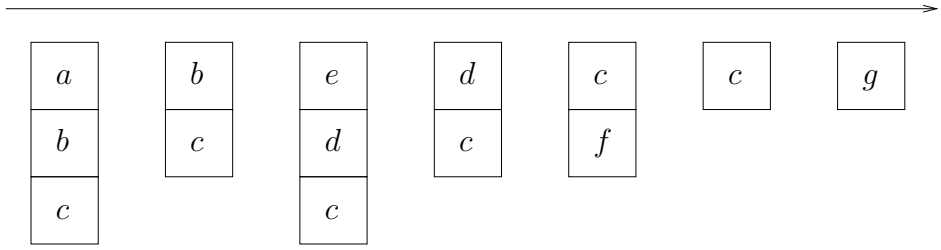


FIG. 4.4 – Evolution des candidats pour la numérotation LIFO du graphe de la figure 4.1

Algorithme orderHNU **:**
Entrée : Un graphe G
Sortie : φ une numérotation de G
$i \leftarrow 1$
$L(i) \leftarrow$ un sommet sans prédécesseur
Tant que $L(i) \neq \emptyset$
  $P \leftarrow$ les prédécesseurs des sommets de $L(i)$
    qui n'ont pas encore été numérotés
  si $P = \emptyset$
    $\varphi(u) \leftarrow i$
    $i \leftarrow i + 1$
    $(L(i) \leftarrow L(i-1) - \{u\}) \cup \Gamma^+(u)$
  sinon
    $L(i) \leftarrow L(i) \cup P$
  finsi
fintantque
finalgo

TAB. 4.4 – Heuristique HNU

termine car G est supposé connexe. Le second cas est traité au plus $n-1$ fois car il ajoute au moins un sommet à la liste des candidats. De plus, à chaque itération les prédécesseurs du sommet choisi sont parcourus, donc le nombre d'opérations est au plus $\mathcal{O}(nm)$. Pour des applications pratiques, le degré des sommets est borné, ce qui mène à une complexité de $\mathcal{O}(n^2)$.

4.1.4 Heuristique ITC

Nous avons aussi développé des heuristiques ayant la propriété d'atteindre des numérotations optimales pour certaines classes de graphes [BMKS03]. Par exemple, l'algorithme montré au tableau 4.5 est optimal pour les anti-arborescences (ITC, pour *InTree Com-*

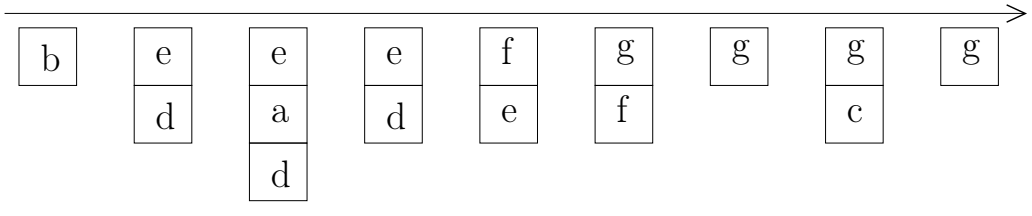


Fig. 4.5 – Evolution des candidats pour la numérotation HNU du graphe de la figure 4.1

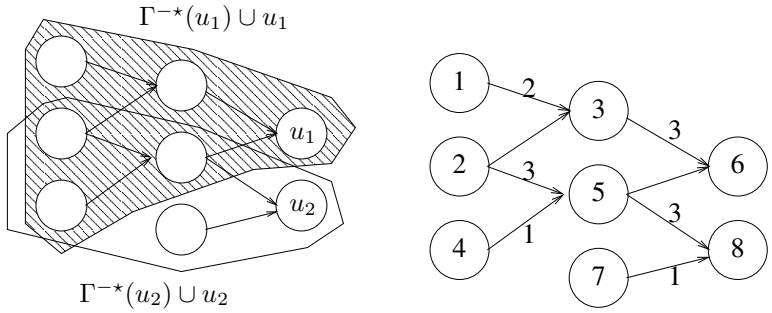
fonction orderITC(T,i) **:int**
si $T \neq \emptyset$ **alors**
  on suppose que $T = \{u_1, \cdots, u_N\}$,
    où $|\Gamma^{\star -}(u_1)| \geq \cdots \geq |\Gamma^{\star -}(u_N)|$
  pour $k = 1 \cdots N$ **faire**
    si $\varphi(u_k) = 0$ **alors**
      $i \leftarrow$orderITC($\Gamma^-(u_k)$,i)
      $\varphi(u_k) \leftarrow i$
      $i \leftarrow i + 1$
    finsi
  finpour
finsi
renvoyer i
finfonction

Tab. 4.5 – Heuristique optimale pour les anti-arborescences (ITC)

pliant). Il est appelé avec T =un ensemble contenant tous les sommets sans successeurs et $i = 1$. La fonction de numérotation est initialisée à $\varphi(u) = 0$ pour tout sommet u de G.

La complexité de l'heuristique ITC est plus importante que celle de l'heuristique HNU, puisque pour chaque sommet on doit calculer la fermeture transitive pour la relation de précédence, et trier ses prédécesseurs selon le résultat. Dans les expériences, cette heuristique n'est pas utilisable telle quelle pour cette raison, mais nous souhaiterions l'adapter pour la numérotation de graphes de précédences de systèmes décrits de façon hiérarchique (voir 1), car le nombre de sommets est moindre dans ce cas. Le coût DSC du graphe de la figure 1.5(b) est réduit à 13 par cette heuristique, comme montré en figure 4.6. La structure de ce graphe étant proche d'une anti-arborescence, le résultat est évidemment bon.

Nous montrons dans la section suivante que l'heuristique HNU est efficace pour la numérotation de graphes issus de cas pratiques.



(a) Première étape (b) Résultat final

Fig. 4.6 – Heuristique ITC pour le graphe de la figure 1.5(b)

4.2 Plateforme de tests

Dans cette section, nous présentons les méthodes qui ont été implémentées pour permettre la validation expérimentale de nos modèles. Dans un premier temps (sous-section 4.1), il nous a fallu concevoir et étudier différentes heuristiques destinées à résoudre les problèmes minDSC et minUCS. Afin de pouvoir réaliser des études statistiques, nous avons aussi implémenté (sous-section 4.2.1) un générateur simple de graphes pseudo-aléatoires. Enfin (sous-section 4.2.2), nous détaillons la plateforme de simulation, c'est-à-dire le processus qui nous permet dans la suite de valider les modèles et les meilleures heuristiques.

4.2.1 Générateur de graphes pseudo-aléatoires

Pour certaines mesures de statistiques, un générateur pseudo-aléatoire de graphes a aussi été conçu. La méthode est simple : n sommets sont numérotés de 1 à n, et m arcs (u, v) sont ajoutés de façon pseudo-aléatoire, en suivant une distribution uniforme. L'absence de circuit est assurée par la condition d'ajout qui impose $u < v$. Des méthodes plus complexes existent pour générer de meilleurs graphes au sens aléatoire du terme, mais celle que nous avons choisie est suffisante puisque nous n'analysons pas de statistiques complexes sur les résultats.

4.2.2 Simulation

Les méthodes qui sont habituellement appliquées pour évaluer la qualité d'une heuristique sont souvent difficile à utiliser dans le cas des problèmes de numérotation de graphes, pour la raison qu'il est difficile d'obtenir de bonnes bornes, quel que soit le critère. Pour cette raison, et aussi parce que c'est la méthode utilisée en général dans les simulateurs, nous avons décidé de comparer nos numérotations avec celles obtenues par une numérotation Random. Cette comparaison a pu être réalisée grâce à l'élaboration d'un environnement simplifié de simulation décrit en figure 4.7.

Tout d'abord, un graphe de précédence est généré à partir d'une description du système à simuler. A partir du graphe de précédence, un programme en langage C est généré. Ce programme est ensuite compilé à l'aide du compilateur `gcc` (avec ou sans option d'optimisation), et les temps d'exécution des programmes sont mesurés. Les scores DSC et UCS sont aussi calculés. Le processus est résumé en figure 4.7.

Jusqu'à une taille suffisante du graphe, deux facteurs perturbent les résultats et empêchent une bonne analyse des résultats. D'une part, quand les temps d'exécution sont trop faibles, les mesures sont difficiles. D'autre part, quand le faible nombre de variables garantit que l'exécution peut être faite intégralement dans des bas niveaux de mémoire, nos optimisations n'offrent que peu d'intérêt. Pour cette raison, les mesures ont été réalisées pour des graphes contenant au moins 10^3 sommets.

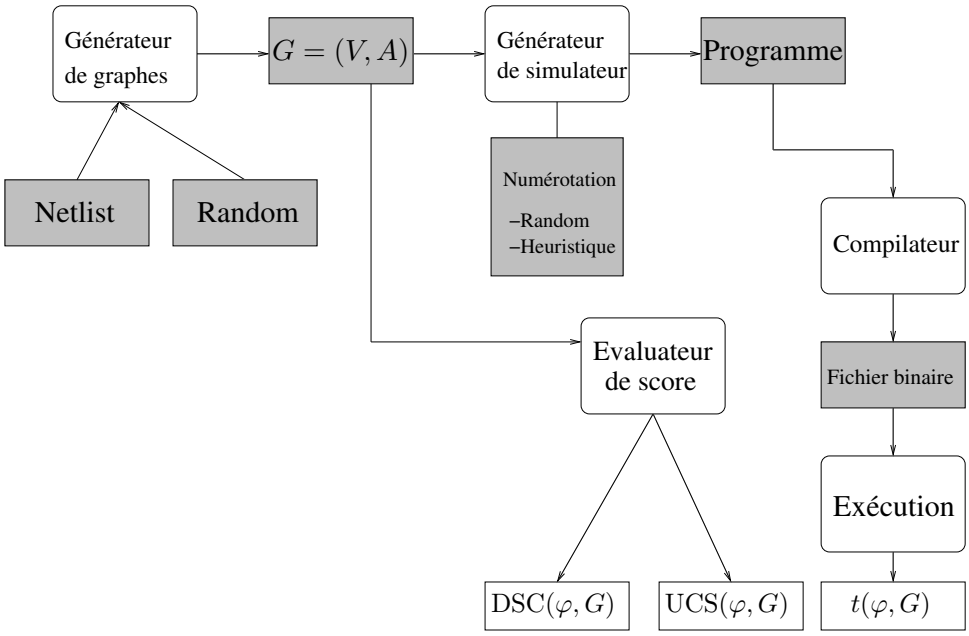


Fig. 4.7 – Processus de validation des modèles

4.3 Performances des heuristiques

4.3.1 Comparaison des différentes heuristiques

Les résultats des différentes heuristiques sont présentés dans la figure 4.8. Le mode opératoire est le suivant. Le graphe est généré à partir d'un diviseur IEEE 64 bits. On génère pour les heuristiques FIFO, LIFO et celle précédemment décrite une numérotation, puis un exécutable. Ensuite, on mesure les durées de 100 exécutions afin d'éliminer le bruit de fond causé par le système. Les chiffres pour l'heuristique RANDOM sont des moyennes pour 10 numérotations. On remarque que les différentes options de compilation de `gcc` n'influencent pas les temps d'exécution, mais en revanchent influencent grandement les temps de compilation (un facteur 100 entre `-O0` et `-O3`).

Pour chaque heuristique testée, trois points sont représentés : le temps d'exécution minimal observé, le temps moyen et le temps maximal.

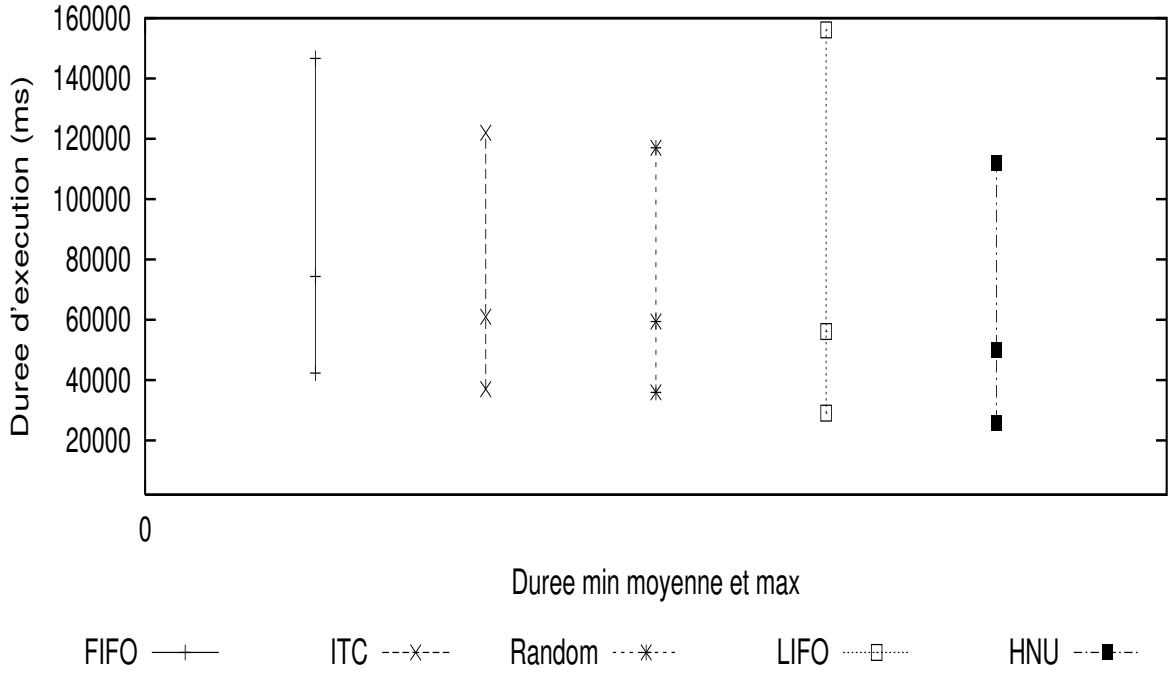


Fig. 4.8 – Comparaison des différentes heuristiques (100 exécutions pour un graphe aléatoire de 10^6 sommets)

4.3.2 Accélération du temps d'exécution

Afin de valider notre méthode, deux éléments sont importants. Tout d'abord, les modèles représentent-ils correctement le temps d'exécution ? Une diminution des scores DSC et UCS par une méthode quelconque conduit-elle bien à une diminution du temps d'exécution ? C'est seulement dans un second temps que l'intérêt de la mise en œuvre de méthodes destinées à diminuer les scores prend un sens. Le deuxième élément des expériences est alors de vérifier que l'heuritique HNU conduit bien à des numérotations de score plus faible que les autres heuristiques.

Nous vérifions que la meilleure des heuristiques conduit bien à des diminutions des temps d'exécution pour la simulation de graphes issus de composants réels. Le tableau 4.6 montre les améliorations des temps d'exécution obtenue grâce à notre heuristique en comparaison avec une numérotation pseudo-aléatoire. Les graphes sur lesquels les résultats on été établis sont les graphes de précédences d'opérateurs arithmétiques générés à l'aide de GenOptim [Hou97] : un diviseur IEEE 64 bits, un multiplieur IEEE 64 bits. Bien que l'évaluation des coûts soit déterministe, les durées d'exécution affichées sont en réalité des moyennes. En effet, à cause du bruit du système d'exploitation sous lequel fonctionne le programme, on a observé des variances pouvant atteindre le double de la valeur moyenne. Empiriquement, la convergence est généralement atteinte après 100 itérations.

Les valeurs pour les graphes générés aléatoirement sont des moyennes. 10 graphes du nombre de sommets et d'arcs correspondant ont été générés. Les scores DSC et UCS représentés pour l'heuristique HNU sont les moyennes de leurs scores. Pour chaque graphe, 70 numérotations Random ont été générées, et les moyennes des scores UCS et DSC calculées. Là encore, les temps d'exécutions sont des moyennes sur 100 simulations.

Il a été observé lors de ces expériences que pour un grand nombre de graphes générés pseudo-aléatoirement, ainsi que pour les graphes générés à partir de composants réels, les heuristiques que nous avons élaborées améliorent les scores DSC et UCS par rapport aux numérotations pseudo-aléatoires. Les résultats du tableau 4.6 illustrent ce propos, qui est aussi vrai pour les temps d'exécution moyens.

4.4 Conclusion

Ce travail a démontré que l'étude de modèles théoriques simples de gestion de la mémoire peut mener à des améliorations de durées d'exécution de programmes en pratique. En effet, les heuristiques que nous avons proposées permettent le réarrangement efficace d'un code de simulation avant sa compilation. Les prochaines étapes de ces travaux sont d'intégrer ces heuristiques dans un environnement de simulation réel.

Lors de nos travaux, nous avons en fait intégré l'heuristique HNU dans un tel en-

		Diviseur	Multiplieur	Graphe aléatoire
Sommets		35078	3109	3000
Arcs		76023	7923	6000
UCS	aléatoire	82252692	1515454	1522150
score	heuristique	80221903	1318562	1452577
Amélioration		2%	13%	5%
DSC	Aléatoire	68394731	883191	887046
score	heuristique	66694305	750756	845254
Amélioration		2.5%	15%	5%
Durée	Aléatoire	59461	7972	8438
d'exécution (ms)	heuristique	49952	6428	7810
Accélération		16%	19.5%	7%

Tab. 4.6 – Accélération des temps d'exécution entre l'heuristique HNU et Random

vironnement (simulateur systemC), sans résultats probants. En effet, l'observation des exécutions du simulateur a montré que le temps effectif passé à la simulation des parties combinatoires des circuits représentent une faible partie du temps total de simulation, au contraire de l'ordonnancement globale des différents éléments des systèmes. Or, c'est justement sur ces parties combinatoires que portent les améliorations amenées par notre méthode. Nos futures expériences sont donc destinées à la porter dans des simulateurs de systèmes dans lesquels la partie combinatoire est prépondérante, tels par exemple les simulateurs de composants arithmétiques.

Conclusions et perspectives

Conclusions

Dans cette thèse, nous avons introduit un nouveau critère d'optimisation pour les problèmes de numérotation de graphes orientés : le critère *Uniform Cost Stack*. Nous avons montré que la minimisation de ce critère est un problème difficile dans le cas des graphes de profondeur 2, mais qu'une numérotation optimale peut être atteinte en un nombre polynomial d'opérations dans le cas des arborescences et des anti-arborescences.

Nous avons aussi étudié la généralisation aux graphes orienté d'un problème classique de numérotations de graphes : Sumcut. Ces travaux nous ont permis de montrer que le problème *Minimum Directed Sumcut* est difficile dans le cas général. En revanche, nous avons montré plusieurs classes de graphes orientés pour lesquelles les sous-problèmes sont résolvables en temps polynomial : les anti-arborescences (voir 3.2.1), les arborescences (voir 3.2.2), les graphes série-parallèles (voir 3.2.3), et les ordres d'intervalles (voir 3.2.4).

Un des apports de nos travaux est aussi d'avoir montré que, à partir de la description d'un système, ces modèles permettent de donner une estimation de la durée d'exécution du programme de simulation au cycle près sur un ordinateur, ces coûts représentant une estimation des durées de chargement des variables du programme à partir de la mémoire. Les études théoriques sur les modèles nous ont permis d'élaborer des heuristiques conduisant à des temps d'exécution inférieurs à ceux qui utilisent des numérotations quelconques.

Perspectives

Les objectifs futurs de nos travaux sont de natures différentes. Tout d'abord, nous aimerions continuer à étudier de nouvelles classes de graphes susceptibles de conduire à des algorithmes polynômiaux pour les problèmes minDSC et minUCS, comme par exemple la classe des ordres quasi-intervalles [Mou03], qui contient à la fois les ordres intervalles et une certaine classe de graphes série-parallèles.. Nous voulons aussi élaborer des modèles à granularité plus importante, de façon à obtenir des algorithmes à plusieurs niveaux, respectant mieux la description hiérarchique des systèmes.

Au niveau des expériences, nous voulons intégrer complètement nos heuristiques dans un environnement de simulation réel, de manière à pouvoir comparer mieux nos méthodes avec l'existant, et surtout à pouvoir les mettre à disposition des concepteurs de systèmes intégrés. De plus ce type de mise en forme est utile pour soulever de nouvelles problématiques auxquelles ils sont confrontés, et pour lesquelles l'optimisation combinatoire et l'ordonnancement peuvent apporter des réponses.

Plus précisément, nous travaillons actuellement à l'intégration de notre méthode dans un simulateur de composants arithmétiques. En plus de pouvoir la comparer avec les autres méthodes, l'objectif est double. Tout d'abord, ces simulateurs ont la particularité de passer la plus grande partie du temps de simulation dans les parties combinatoires des composants, ce qui est justement le point sur lequel portent nos optimisations. Ceci est à opposer à d'autres simulateurs, qui se focalisent sur un ordonnancement à granularité importante, c'est à dire aux interactions entre les composants du système à simuler. Les optimisations sur les parties combinatoires dans de tels cas ne portent donc que sur une faible partie du temps de simulation, et n'offrent donc que peu d'intérêt. D'autre part, les circuits arithmétiques ont la particularité d'avoir une structure arborescente, à partir d'un certain niveau d'abstraction. Notre méthode étant optimale pour de telles structures, nous étudions la possibilité de numéroter de façon optimale au sens de DSC ou UCS des blocs de sommets. Dans le cas où ces blocs contiennent un faible nombre de sommets, nos modèles de mémoire gardent un sens, et nous attendons des améliorations de temps de simulation meilleures que celles obtenues avec la mise à plat intégrale du circuit. Les travaux préliminaires sur cette méthode ont déjà montré que le problème de la construction de telles partitions de l'ensemble des sommets d'un graphe est un problème non trivial.

Bibliographie

[AH73] D. Adolphson and T.C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25 :403–423, 1973.

[AVL62] G.M. Adelson-Velskii and Y.M. Landis. An algorithm for the organization of information. *Soviet Moth. Dokl.*, 3 :1259–1262, 1962.

[Bak74] Kenneth R. Baker. Introduction of sequencing and scheduling. Technical report, Wiley & Sons, August 1974.

[BGT98] Bodlaender, Gustedt, and Telle. Linear-time register allocation for a fixed number of registers. In *SODA : ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1998.

[BMKS03] T. Bossart, A. Munier-Kordon, and F. Sourd. Two models for the optimization of integrated circuit simulators. Technical report, Laboratoire d'Informatique de Paris VI, 2003. accepted in Discrete Applied Mathematics, `http ://asim.lip6.fr/∼bossart/`.

[Bru95] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.

[Bru98] P. Brucker. *Scheduling algorithms*. Springer, Berlin, 1998.

[CP95] Philippe Chrétienne and Christophe Picouleau. *Scheduling Theory and its Applications*, chapter 4, pages 65–90. Wiley & Sons, Laboratoire LITP, Université Pierre et Marie Curie, 1995.

[DeV97] Charles J. DeVane. Efficient circuit partitioning to extend cycle simulation beyond synchronous circuits. In *ICCAD '97 : Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pages 154–161, Washington, DC, USA, 1997. IEEE Computer Society.

[DGPT91] J. Díaz, A.M. Gibbons, M.S. Paterson, and J. Torán. The minsumcut problem. *Lecture Notes in Computer Science*, 519 :65–79, 1991.

[Dia79] J. Diaz. The delta-operator. *Fundamentals of Computation Theory*, pages 105–111, 1979.

[DPS02] Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3) :313–356, 2002.

[FLLO95] Robert S. French, Monica S. Lam, Jeremy R. Levitt, and Kunle Olukotun. A general method for compiling event-driven simulations. In *DAC '95 : Proceedings of the 32nd ACM/IEEE conference on Design automation*, pages 151–156, New York, NY, USA, 1995. ACM Press.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[Gol04] M.C. Golumbic. *Algorithmic Theory and Perfect Graphs*. Elsevier, 2004.

[Gos93] J. B. Gosling. *Simulation in the Design of Digital Electronic Systems*. Cambridge University Press, 1993.

[Han93] Jim Handy. *The cache memory book*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[Hom01] D. Hommais. *Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel-logiciel*. PhD thesis, University of Paris VI, 2001. In French.

[Hou97] A. Houelle. *GenOptim : un environnement d'aide à la conception de générateurs de circuits portables optimisés en performance et en surface*. PhD thesis, University of Paris VI, 20 June 1997. In French.

[Jen91] Glenn Jennings. A case against event-driven simulation for digital system design. In *Proceedings of the 24th annual symposium on Simulation*, pages 170–176. IEEE Computer Society Press, 1991.

[Kar93] Richard M. Karp. Mapping the genome : some combinatorial problems arising in molecular biology. In *STOC '93 : Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 278–285, New York, NY, USA, 1993. ACM Press.

[Ken69] D.G. Kendall. Incidence matrices, interval graphs, and seriation in archaeology. *Pacific J. Math*, pages 565–570, 1969.

[LY94] Y. Lin and J. Yuan. Profile minimization problem for matrices and graphs. *Acta Math. Appl. Sinica, English-Series*, pages 107–112, 1994.

[Moo65] Gordon Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[Moo75] Gordon Moore. Progress in digital integrated electronics. In *1975 IEEE International Electronic Devices meeting*, 1975.

[Mou03] Aziz Moukrim. Scheduling unitary task systems with zero-one communication delays for quasi-interval orders. *Discrete Appl. Math.*, 127(3) :461–476, 2003.

[Pug91] William Pugh. Uniform techniques for loop optimization. In *ICS '91 : Proceedings of the 5th international conference on Supercomputing*, pages 341–352, New York, NY, USA, 1991. ACM Press.

[PY79] Christos H. Papadimitriou and Mihalis Yannakakis. Scheduling interval-ordered tasks. *SIAM J. Comput.*, 8(3) :405–409, 1979.

[RAK91] R. Ravi, Ajit Agrawal, and Philip N. Klein. Ordering problems approximated : Single-processor scheduling and interval graph completion. In *ICALP*, pages 751–762, 1991.

[RD05] Mehrdad Reshadi and Nikil Dutt. Generic pipelined processor modeling and high performance cycle-accurate simulator generation. In *DATE '05 : Proceedings of the conference on Design, Automation and Test in Europe*, pages 786–791, Washington, DC, USA, 2005. IEEE Computer Society.

[Rob03] A. Robert. Modélisation, analyse et optimisation d'un simulateur de circuits. Master's thesis, University of Paris VI, 2003. in French.

[Row94] James A. Rowson. Hardware/software co-simulation. In *31 ST ACM/IEEE Design Automation Conference*. ACM Press, 1994.

[SBR05] Jurgen Schnerr, Oliver Bringmann, and Wolfgang Rosenstiel. Cycle accurate binary translation for simulation acceleration in rapid prototyping of socs. In *DATE '05 : Proceedings of the conference on Design, Automation and Test in Europe*, pages 792–797, Washington, DC, USA, 2005. IEEE Computer Society.

[Sch95] L.A.M. Schoenmakers. A new algorithm for the recognition of series parallel graphs. Technical Report CS-R9504, Centrum voor Wiskunde en Informatica, january 1995.

[Set75] R. Sethi. Complete register allocation problems. *SIAM J. Computing*, 4(3) :226–248, September 1975.

[Smi56] W.E. Smith. Various optimizers for single stage production. *Naval Research Logistics Quarterly*, **3** :59–66, 1956.

[SS92] Emily J. Shriver and Karem A. Sakallah. Ravel : assigned-delay compiled-code logic simulation. In *ICCAD '92 : Proceedings of the 1992 IEEE/ACM international conference on Computer-aided design*, pages 364–368, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[ST97] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection ? *SIAM Journal on Scientific Computing*, 18(5) :1436–1445, 1997.

[Ulr69] Ernst G. Ulrich. Exclusive simulation of activity in digital networks. *Commun. ACM*, 12(2) :102–110, 1969.

[UMR99] Raimund Ubar, Adam Morawiec, and Jaan Raik. Cycle-based simulation with decision diagrams. In *DATE '99 : Proceedings of the conference on Design, automation and test in Europe*, page 95, New York, NY, USA, 1999. ACM Press.

[VTL79] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 1–12. ACM Press, 1979.

[WM90] Zhicheng Wang and Peter M. Maurer. Lecsim : a levelized event driven compiled logic simulation. In *DAC '90 : Proceedings of the 27th ACM/IEEE conference on Design automation*, pages 491–496, New York, NY, USA, 1990. ACM Press.