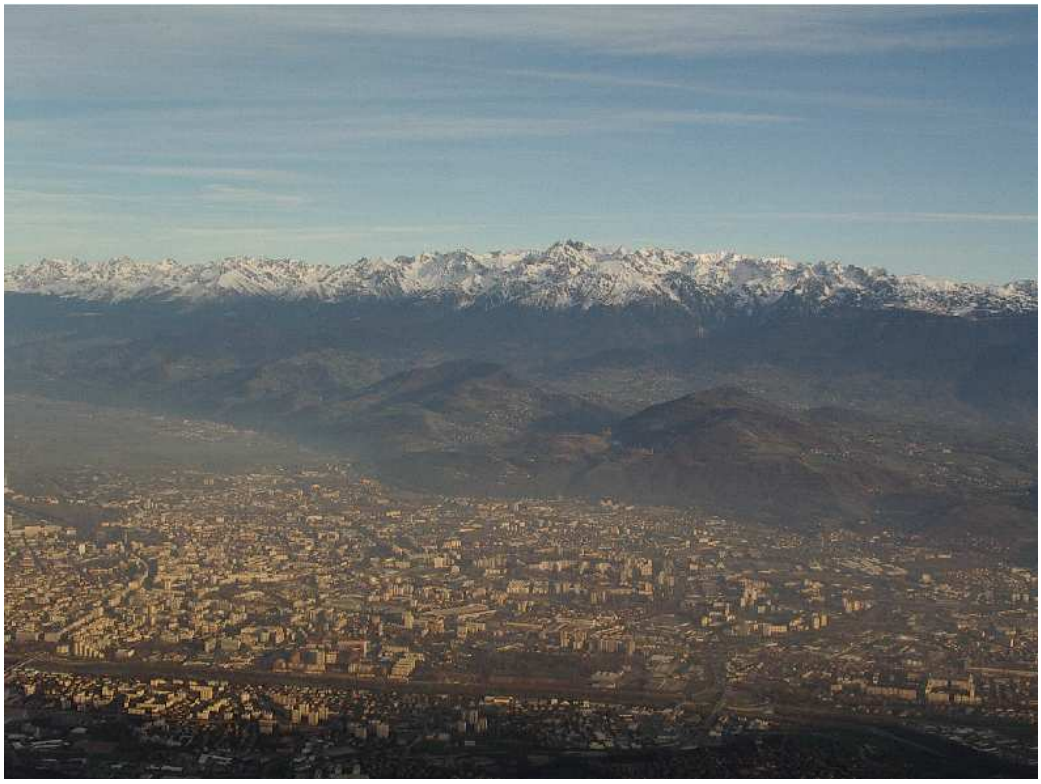
ÉCOLE JEUNES CHERCHEURS
EN
ALGORITHMIQUE
ET
CALCUL FORMEL



Grenoble 29 mars au 2 avril 2004

Table des matières

Avant-Propos

EJCACF 2004 est depuis 1996 la huitième école dans la série des écoles de Jeunes Chercheurs en Algorithmique et Calcul Formel. Cette école concerne principalement les jeunes chercheurs à plus ou moins deux ans de leur thèse dans les domaines de l'algorithmique et du calcul formel.

Les objectifs de cette école sont essentiellement de permettre aux jeunes chercheurs de se rencontrer et de présenter certains de leurs travaux, de compléter leur formation et de leur donner quelques outils permettant une meilleure compréhension de leur discipline.

Les thèmes de l'EJCACF 2004 sont ceux du pôle "Algorithmique et Calcul Formel" du GDR ALP, et la plupart se situent à l'interface naturelle entre mathématiques et informatique :

- Aspects algorithmiques de l'algèbre, de la géométrie, de la théorie des nombres, théorie des Graphes.
- Calcul formel et applications.
- Théorie algorithmique de l'information.
- Analyse d'algorithmes et algorithmique probabiliste.
- Codage et cryptographie.
- Objets graphiques pour les mathématiques et l'informatique.
- Algorithmique du texte et du génome.
- Systèmes à événements discrets et automates, systèmes dynamiques, systèmes hybrides.
- Arithmétique des ordinateurs.
- Géométrie algorithmique.
- Algorithmique combinatoire.

En vous souhaitant une excellente lecture,

Mars 2004

Le comité d'organisation

Comité scientifique

Jean-Guillaume Dumas, MC à l'UJF, LMC
Jean-Guillaume.Dumas@imag.fr

Dominique Duval, Professeur à l'UJF, LMC
Dominique.Duval@imag.fr

Christiane Frougny, Professeur à Paris 8, LIAFA
Christiane.Frougny@liafa.jussieu.fr

Jean-Michel Muller, DR CNRS, ENS Lyon, LIP
Jean-Michel.Muller@ens-lyon.fr

Natacha Portier, MC à l'ENS Lyon, LIP
Natacha.Portier@ens-lyon.fr

Comité d'organisation

Jean-Guillaume Dumas, MC à l'UJF, LMC

Natacha Portier, MC à l'ENS Lyon, LIP

Page web
Etienne Farcot, doctorant INPG, LMC

Gestion financière
Corinne Kabsch, AI LMC-CNRS

Affiche et ski
Clément Pernet, doctorant UJF, LMC

Support de cours et exposés
Aude Rondepierre, doctorant INPG, LMC

Déroulement de l'école

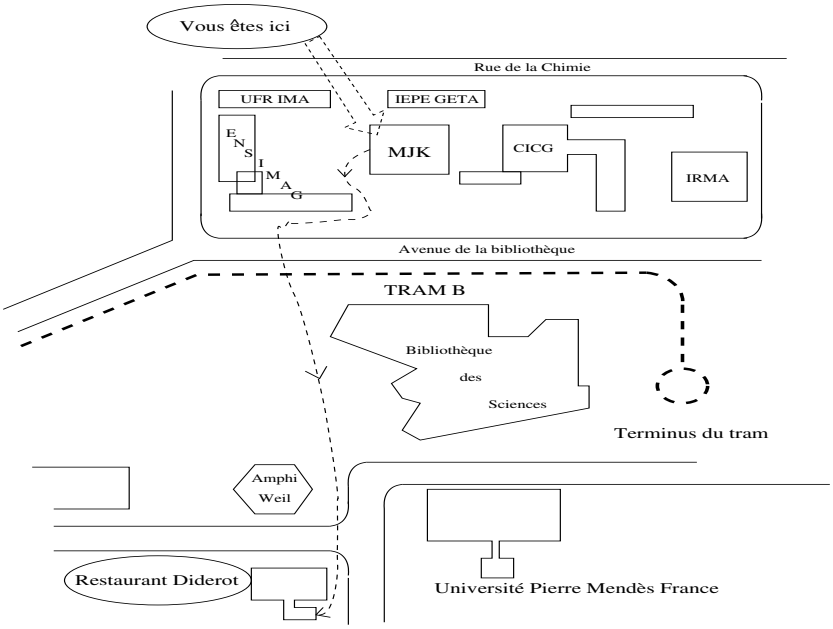
Cours et exposés :

Chaque journée de la semaine est consacrée à l'une de ces thématiques. Les matinées sont consacrées aux cours magistraux (cinq cours de 4 heures) dans les thématiques spécifiques de cette année et les après-midi aux exposés des jeunes chercheurs.
Les cours et exposés se dérouleront à la

Maison Jean Kuntzmann
110 avenue de la Chimie
Domaine Universitaire
Saint Martin d'Hères - France

Repas de midi

Les repas seront pris au restaurant universitaire Diderot, salle Joël Duthérian.



Pour aller au restaurant universitaire Diderot, salle Joël Duthérian

Planning de la semaine

Le planning est le suivant :

Lundi 29 mars	
Optimisation Combinatoire	
8h45-9h00	Accueil
9h00-10h45	**M. Demange**
10h45-11h00	Café
11h00-12h45	**M. Demange**
13h00-14h30	Déjeuner
14h30-14h50	A. Tsiry
14h50-15h10	A. Syrine
15h10-15h30	C. Martig

Mardi 30 mars	
Parallélisme et systèmes complexes	
8h45-9h00	Accueil
9h00-10h30	**J.-L. Roch**
10h30-10h45	Café
10h45-11h45	**D. Trystram**
11h45-12h45	**D. Naddef**
13h00-14h30	Déjeuner
14h30-14h50	Y. Slama
14h50-15h10	I. Nafkha
15h10-15h30	G. Jean-Loup

Mercredi 31 mars	
Algorithmes pour l'image de synthèse	
8h45-9h00	Accueil
9h00-10h45	**X. Décoret**
10h45-11h00	Café
11h00-12h45	**C. Soler**
13h00-14h30	Déjeuner
14h30-14h50	S. Graillat
14h50-15h10	G. Cheze
15h10-15h30	A. Mahboubi

Jeudi 1 avril	
Arithmétique des ordinateurs	
8h45-9h00	Accueil
9h00-10h30	**A. Tisserand**
10h30-10h45	Café
10h45-11h45	**N. Revol**
11h45-12h45	**P. Zimmermann**
13h00-14h30	Déjeuner
14h30-14h50	D. Stehle
14h50-15h10	P. Giorgi
15h10-15h30	C. Pernet

Vendredi 2 avril	
Systèmes hybrides	
8h45-9h00	Accueil
9h00-10h45	**A. Girard**
10h45-11h00	Café
11h00-12h45	**T. Dang**
13h00-14h30	Déjeuner
14h30-14h50	E. Farcot
14h50-15h10	L. Tournier
15h10-15h30	A. Rondepierre

Soutiens

Université Joseph Fourier, Grenoble
http ://www.ujf-grenoble.fr/

Institut d'Informatique et de Mathématiques Appliquées de Grenoble
http ://www.imag.fr/

Centre National de la Recherche Scientifique
http ://www.cnrs.fr/

Algorithmique, Langage et Programmation
http ://www.liafa.jussieu.fr/%7Ealp/

Région Rhône-Alpes
http ://www.cr-rhone-alpes.fr/

Institut National de Recherche en Informatique et en Automatique
http ://www.inria.fr/

Années précédentes

Les sites des années précédentes :

1996 Nice http ://www.essi.fr/ecole-AMI/

1997 Marseille, http ://iml.univ-mrs.fr :80/∼ami/

1999 Bordeaux, http ://www.labri.fr/Perso/∼weil/ecolegdr/appel.html

2000 Caen, http ://www.greyc.unicaen.fr/anciens-evenements/ejc2000/

2001 Lyon, http ://www.ens-lyon.fr/∼nportier/ejc2001/

2002 Lille, http ://www.lifl.fr/∼ejc2002/

2003 Marne la Vallée, http ://www-igm.univ-mlv.fr/∼ejc2003/

2004 Grenoble, http ://www-lmc.imag.fr/EJCACF2004/

Conférences

Cette année, les axes thématiques sont :

⋆ Systèmes hybrides (Jean-Guillaume Dumas, Université Joseph Fourier, Grenoble)

⋆ Programmation parallèle et systèmes complexes (Jean-Louis Roch, ENSIMAG, Grenoble)

⋆ Algorithmes pour l'image de synthèse (Gilles Debunne, CNRS, Grenoble)

⋆ Arithmétique des ordinateurs (Nathalie Revol, ENS Lyon)

⋆ Optimisation Combinatoire (Marc Demange, ESSEC, Paris)

Optimisation Combinatoire

Responsable : Marc Demange

Conférencier : Marc Demange

Sommaire

Algorithmes d'approximation : un petit tour en compagnie d'un voyageur de commerce*

Marc Demange
ESSEC

demange@essec.fr

Résumé

Cet article est une introduction à l'approximation polynomiale de problèmes NP-difficiles. Il présente une sélection de résultats récents dans le domaine et représentatifs d'une large gamme de techniques, de résultats et de problématiques de l'approximation. L'objectif est de présenter ce domaine au lecteur non spécialiste d'optimisation combinatoire. Pour cela, ont été sélectionnés des résultats simples ou présentés de manière simplifiée afin de pouvoir présenter la preuve de la plupart d'entre eux. Entrer dans ces preuves est en effet la meilleure façon de s'imprégner du domaine. Toujours par souci de simplicité et d'unité, tous les résultats développés dans cet article concernent le même problème, à savoir, le problème de voyageur de commerce, sous différentes variantes bien entendu. Il ne s'agit nullement d'un survey, les résultats proposés n'ayant aucun caractère exhaustif.

Sont présentés essentiellement des analyses d'algorithmes d'approximation sous deux points de vue complémentaires associés à deux mesures d'approximation. Toutefois, ces résultats seront l'occasion d'esquisser les facettes du domaine qui ne sont pas traitées ainsi que ses enjeux.

Enfin, j'espère, à titre personnel, vous faire partager par ces quelques pages un domaine que je trouve fascinant.

Avertissement

Dans le texte qui suit, le terme *voyageur de commerce* est pris dans son sens mathématique. N'y voyez donc aucune persécution d'une corporation que je respecte pleinement.

1 Introduction

L'*approximation polynomiale* puise ses fondements dans la découverte que certains problèmes d'optimisation combinatoire - pourtant d'expression très simples puisqu'ils consistent à op-

*Cet article sera publié en 2004 dans la *Gazette des mathématiciens*, Société mathématique de France.

timiser une fonction $f : 2^{\{1,\ldots,n\}} \rightarrow I\!N$ évaluable en temps polynomial - pourraient ne pas admettre d'algorithme de résolution polynomial [1]. L'optimiseur devrait alors se contenter d'un algorithme décrivant, plus ou moins complètement, l'ensemble $2^{\{1,\ldots,n\}}$ de toutes les solutions possibles afin de comparer les valeurs correspondantes de f. Le conditionnel correspond ici la fameuse conjecture P≠NP dont les origines remontent environ au début des années 1960 [18] et qui a pris son plein envol avec la « découverte » du premier problème NP-complet par cook en 1971 [12]. Au sein de la classe des problèmes de NP (nous aurons l'occasion d'y revenir dans la partie 2), les problèmes *NP-complets* sont vite devenus le symbole d'une difficulté intrinsèque les rendant impropres à la « computation ». Face à ce phénomène, l'approximation polynomiale étudie des algorithmes polynomiaux permettant de déterminer, pour des problèmes *NP-difficiles* (en fait version optimisation de problèmes NP-complets) de bonnes solutions, c'est à dire des solutions garantissant, quoi qu'il advienne, un certain niveau de qualité.

L'objectif de cet article est de montrer au lecteur non familier de l'approximation polynomiale quelques exemples suffisamment simples (ou simplifiés) mais néanmoins significatif des différentes facettes de ce domaine. En guise de premier contact avec le domaine, nous aborderons surtout l'analyse d'algorithmes d'approximation et de leurs garanties de performances, pour deux mesures de qualité. Nous évoquerons également, dans un cas très simple, un résultat de difficulté d'approximation pour dégager les différents enjeux de ce domaine. Nous ne pourrons alors évoquer que très brièvement les autres aspects de l'approximation, à savoir notamment l'étude des *réductions en approximation* et la structure de la classe NP en fonction des propriétés d'approximation.

Cet article est une promenade en approximation ... en compagnie d'un voyageur de commerce. En effet, pour ce travail d'initiation, j'ai choisi un problème particulier, le *voyageur de commerce*, qui nous accompagnera sous ses différentes versions tout au long de cette étude. La motivation principale est bien entendu de ne pas ajouter de difficulté par une multiplication des problèmes abordés. Dans le même esprit, j'ai sélectionné quelques résultats significatifs et accessibles sans grande difficulté. Vous êtes donc invités à aborder cet article crayon et brouillon à la main !

Comme il se doit, avant une promenade entre amis, laissez moi vous présenter rapidement le nouvel arrivant : le problème de voyageur de commerce.

1.1 TSP : une histoire déjà longue

À la base du problème se situe la question suivante : *étant données n villes et leurs distances mutuelles, comment les parcourir de sorte que la distance totale soit minimum ?* Si toutes les connexions entre villes sont possibles, il est évident qu'il existe des solutions réalisables ($(n-1)!$ pour être précis). Par contre, si seulement certaines connexions sont pos-

[1] Dont l'exécution nécessite au plus un nombre d'opérations élémentaires polynomial par rapport à la taille n des données.

sibles et si on impose de ne passer qu'une fois et une seule dans chaque ville, l'existence même d'une solution réalisable pose problème : il s'agit du problème d'hamiltonicité [2] connu pour être NP-complet. Le problème de voyageur de commerce est donc intimement lié au problème de recherche d'un cycle hamiltonien dans un graphe, les sommets du graphe représentant les villes et les arêtes les liens entre les villes, i.e. la possibilité d'un passage directe de l'une à l'autre.

W. R. Hamilton a donné son nom à de tels cycles pour avoir étudié ce problème au milieu du 19 ième siècle, notamment dans le cas de 20 villes situées aux sommets d'un dodécaèdre régulier : le « voyage autour du monde » et le « jeu icosien », (bien que posé sur un dodécaèdre), commercialisé vers 1859, sont différentes représentations de ce problème.

Jeu icosien (source [37])

Mais l'origine du problème hamiltonien remonterait à un siècle auparavant lorsqu'Euler et Vandermonde se sont demandé comment un cavalier pourrait visiter toutes les cases d'un échiquier. Pour plus de détails sur ces deux exemples, on pourra se reporter notamment à [8], ch. 10 et [26], ch. 1.

Quant au problème de voyageur de commerce, dans sa version pondérée, sa première mention remonterait à un livre allemand (1832), avec un exemple sur 45 villes. Mais la première formulation mathématique du problème est attribuée à Menger, vers 1930.

Le lecteur intéressé par la fascinante histoire de ce voyageur pourra se reporter notamment à [26], ch. 1 et à [36].

1.2 TSP : séquence fascination

Le succès de ce problème, qui a suscité et suscite encore de très nombreux travaux, tient d'abord à sa formulation simple, en forme de défi. Bien d'autres problèmes combinatoires, non moins populaires, comme les problèmes de coloration par exemple, inspirent la même fascination de la complexité derrière un visage d'enfant. Mais **TSP** a quelque chose de particulier

[2] Consiste à décider si un graphe contient un cycle hamiltonien, i.e. visitant une fois et une seule chaque sommet.

au point, notamment, d'inspirer depuis de nombreuses années des concours internationaux, une sorte de course sans fin à qui résoudra des instances de plus en plus volumineuses [37].

Toutefois, comme pour la plupart des problèmes combinatoires, derrières des formulations en forme de jeux mathématiques se cachent de nombreuses applications potentielles. **TSP** n'y échappe pas. On pense évidemment à des problèmes de tournées qui ne sont pas si éloignés du personnage mythique qui part sillonner les routes son attaché-case à la main. Mais le problème a de nombreuses autres applications plus inattendues telle que la fabrication de circuits intégrés, des problèmes de découpe, d'ordonnancement ou encore d'arrangement des lignes et colonnes de tableaux, des applications en génomique, pour la conception de réseaux de fibre optiques, ... (pour plus de détails, se reporter à [26], ch. 2 ou également [37]).

1.3 TSP : la famille

Pour conclure cette présentation, voici la famille **TSP**. Une instance du problème général se formule ainsi :

> **Min TSP** : On se donne un graphe complet (non orienté pour le cas qui nous intéresse) à n sommets (K_n) arêtes-valué. L'ensemble des sommets est noté $\{1, \ldots, n\}$. Pour chaque arête e, on note $c(e)$ sa valeur (en fait un coût). Un cycle hamiltonien (ou tour lorsqu'aucune confusion n'est possible) est une permutation des sommets $T = (v_1, v_2, \ldots v_n)$ et sa valeur est $c(T) = \sum_{i=1}^{n-1} c(v_i v_{i+1}) + c(v_n v_1)$. Le problème consiste alors à déterminer un tour de valeur minimum.

De nombreuses variantes existent en fonction du système de coûts adopté. Dans le cas général, c est une fonction $c : \{1, \ldots, n\} \to \mathbb{Q}$ avec comme simple restriction de pouvoir être évaluée en temps polynomial. Une restriction naturelle consiste à imposer à c de vérifier les inégalités triangulaires. Le problème de *voyageur de commerce métrique*, noté $\mathbf{\Delta - TSP}$, correspond au cas où c est une distance. Un cas particulier de $\mathbf{\Delta - TSP}$ consiste à restreindre les valeurs d'arêtes à 1 ou 2 : on le note $\mathbf{TSP}_{1,2}$. Enfin, une autre restriction, qui correspond au cadre des formulations d'origine, concerne le cas où c est une distance euclidienne, les villes étant des points de $\mathbb{R}^p$. Nous appelons ce cas *voyageur de commerce euclidien* et le notons $\boldsymbol{\ell_2 - \text{Min TSP}}$. Dans cet article, nous envisageons ces différentes versions et mettrons en évidence combien le système de valuation conditionne les possibilités de résolution efficace.

Il existe évidemment de très nombreux cousins qui sont définis en modifiant légèrement les contraintes et l'objectif : tous les problèmes évoqués ont leur version maximisation, dans certains cas, le voyageur ne rentre pas chez lui ou encore doit se répartir les villes avec des confrères (très utile dans les modèles de tournées). Plusieurs généralisations imposent également des contraintes temporelles sur les sommets qui sont vus alors comme des requêtes à servir : à chacune est associée une fenêtre de temps au cours de laquelle elle peut être servie avec, éventuellement, une durée pour servir une requête. Mentionnons enfin la version dans laquelle le voyageur se fixe un objectif de p villes parmi n et s'en contente. Je laisse le soin à chacun d'imaginer d'autres versions et/ou de se plonger dans la littérature très fournie

autour de ce problème. Le site internet [37] permet de trouver une première base de références bibliographiques.

2 Voyageur de commerce : un métier difficile

Min TSP figure sans doute parmi les plus célèbres problèmes de la (désormais très vaste) classe des problèmes *NP-difficiles*. L'une des conséquences intuitives les plus populaires est que, jusqu'à preuve du contraire (i.e. si P≠NP), aucun algorithme polynomial ne peut résoudre ce problème. Ainsi, le paradigme « poétiquo-naïf » dont (entre autres) la Recherche Opérationnelle est particulièrement friande place le métier de voyageur de commerce au rang des activités les plus difficiles au même titre que proviseur de lycée la veille de la rentrée (ordonnancement), postier en milieu rural ou s'il est chinois et à moitié désorienté (chinese postman in mixed graphs [18]), campeur (problème de sac à dos), ...

2.1 La classe NP-complet (NP-C) : un bref aperçu

Pour caractériser la difficulté de ce problème, la démarche la plus usuelle est d'abord de le lier à la classe des problèmes *NP-complets*. Si on modélise un problème de décision (réponse oui ou non) comme « décider si un mot donné (instance) appartient à un langage (problème) », alors NP désigne la classe des problèmes de décision qui peuvent être résolus par un algorithme (machine de Turing) non-déterministe polynomial. La classe NP-complet contient les problèmes les plus difficiles de NP en ce sens que la résolution de l'un d'entre eux par un algorithme (déterministe) polynomial entraînerait la possibilité de résoudre en temps polynomial tout problème de NP, rendant ainsi une machine de Turing non déterministe polynomialement équivalente à une machine déterministe. La notion de réduction polynomiale entre deux problèmes de décision permet d'exprimer cette difficulté de manière plus précise : une réduction polynomiale $\propto$ de Π_1 à Π_2 permet de construire en temps polynomial, pour toute instance I_1 de Π_1 de taille n_1 une instance $\propto (I_1) = I_2$ de Π_2 de taille n_2 qui est une instance positive (réponse oui) si et seulement si I_1 est positive. Le caractère polynomial de la réduction permet en particulier de déduire qu'il existe un polynôme P indépendant de I_1 tel que $n_2 \leq P(n_1)$. Par conséquent un algorithme polynomial pour Π_2 peut immédiatement être transformé via la réduction en un algorithme polynomial pour Π_1 ; c'est en ce sens que Π_2 peut alors être considéré comme au moins aussi difficile que Π_1. Un problème *NP-complet* est alors un problème de NP auquel se réduit polynomialement tout problème de NP. Le problème de satisfiabilité fut le premier problème NP-complet mis en évidence par Cook en 1971 [12]. Il s'agit, étant donné une collection de clauses disjonctives sur n variables booléennes, de décider s'il existe une affectation de vérité aux variables rendant toutes les clauses vraies. Depuis ce « premier problème NP-complet », de très nombreux problèmes ont été identifiés comme tels. Dans la quasi totalité des cas, l'appartenance à la classe NP-complet est prouvée par réduction : un problème de NP auquel se réduit polynomialement un problème NP-complet

est lui-même dans cette classe. Pour de plus amples informations sur le sujet, on pourra se référer notamment à [18, 31].

2.2 Min TSP et NP-C

Parmi les 21 problèmes NP-complets mis en évidence par Karp dans [23] en 1972 figure le problème du *Cycle hamiltonien*, noté **HC**.

> **HC** : Étant donné un graphe, déterminer s'il existe un cycle hamiltonien, c'est à dire passant une fois et une seule en chaque sommet.

La difficulté du problème **Min TSP** s'en déduit immédiatement. Pour se ramener à un problème de décision, on définit la version décisionnelle **Min TSP**$_d$ de **Min TSP** :

> **Min TSP**$_d$: Étant donnés un graphe complet (non orienté pour le cas étudié ici) arêtes-valué et une constante K, existe-t-il un cycle hamiltonien de valeur au plus K, la valeur d'un cycle correspondant à la somme des valeurs de ses arêtes ?

Théorème 2.1 *[18]*
Min TSP$_d$ *est NP-complet.*

Preuve : Cette preuve de NP-complétude compte parmi les plus simples mais permettra au lecteur non familier de ces notions de comprendre les principes d'une telle démonstration.

L'appartenance de **Min TSP**$_d$ à la classe NP est très simple à établir. Il suffit, étant donné une instance, de générer de manière non-déterministe un ordre (avec une complexité linéaire) sur les sommets puis de tester en temps polynomial si cet ordre correspond à un cycle hamiltonien de valeur au plus K. Un tel processus définit une machine de Turing non-déterministe polynomiale. L'instance est positive si et seulement si il est possible de générer une solution dans la phase non-déterministe pour laquelle la phase déterministe reconnaît qu'il s'agit d'un cycle hamiltonien répondant à la question. Ceci correspond exactement ([18]) à la définition de *l'acceptation* dans le modèle des machines de Turing non-déterministe et donc au caractère NP du problème considéré. Intuitivement, un problème de NP s'apparente à une situation de ce type pour laquelle on peut, en temps polynomial, tester si un candidat solution (test) de taille polynomiale répond à la question ; une instance est positive s'il existe une telle solution qui peut alors être générée par la phase non déterministe (dès lors qu'elle peut être représentée en temps polynomial).

Pour montrer l'appartenance à NP-C, on réduit polynomialement **HC** à **Min TSP**$_d$: étant donnée un graphe $G = (V, E)$ d'ordre n, instance de **HC**, on définit le graphe complet sur V et on affecte à l'arête uv le coût 1 si $uv \in E$ et 2 sinon. En posant $K = n$, il est clair que G est hamiltonien si et seulement si il existe dans le graphe complet un cycle hamiltonien de valeur $K = n$ (i.e. ne comprenant que des arêtes de coût 1). ■

La preuve montre en fait que **Min TSP**$_{1,2}$, le problème TSP restreint au cas où les valeurs des arêtes sont dans $\{1, 2\}$, est NP-complet. En particulier, le cas métrique $\Delta-$**Min TSP** est NP-complet. Ceci montre aussi que les restrictions de **Min TSP**, et $\Delta-$**Min TSP** au cas de poids bornés par un polynôme (ou même borné par une constante) sont dans NP-C. En d'autres termes, ces problèmes sont *NP-complets au sens fort* : leur difficulté ne provient pas (seulement) de la taille des pondérations. Le cas euclidien est aussi connu comme NP-complet au sens fort ([18]), mais la réduction, à partir du couplage de dimension 3, est différente.

2.3 NPO : problèmes d'optimisation de NP

Usuellement, la complexité de problèmes d'optimisation est étudiée au travers de la version décisionnelle du problème (**Min TSP**$_d$ dans le cas présent). Ceci n'est pas toujours satisfaisant dès lors qu'on s'intéresse à la résolution de problèmes d'optimisation combinatoire. En effet, la version décisionnelle d'un problème d'optimisation est plutôt liée, du point de vue de la résolution, à la détermination de la valeur optimale et non à la recherche d'une solution optimale. Plus précisément, **Min TSP**$_d$ correspond, pour chaque valeur de K, à comparer K et la valeur optimale de l'instance. Tant que les valeurs réalisables sont des entiers bornés par $2^{p(n)}$ où p est un polynôme en la taille n du problème (ce qui recouvre la quasi totalité des problèmes usuels), il est simple de montrer comment une recherche dichotomique permet d'établir l'équivalence (du point de vue d'une résolution polynomiale) entre la version décisionnelle et le problème d'évaluation consistant à déterminer la valeur optimale. Par contre, l'équivalence algorithmique entre la version évaluation (déterminer la valeur optimale) et la recherche d'une solution optimale est beaucoup moins claire. Elle a été établie ([35], cf. également [6]) lorsque la version décisionnelle est NP-complète mais la preuve n'est pas exploitable du point de vue opérationnel.

Dès lors qu'on s'intéresse à la résolution effective de problèmes d'optimisation, c'est la recherche de solutions qui prend le dessus sur la version décisionnelle. La classe NPO a pour vocation de traiter des problèmes d'optimisation sans référence au problème décisionnel associé. Il s'agit du cadre usuellement adopté pour traiter l'approximation à garanties de performances de problèmes NP-difficiles. La définition explicite de cette classe n'apporterait rien dans cet article, le lecteur intéressé pourra se référer par exemple à [6]. Intuitivement, un problème d'optimisation de NPO comprend une notion de *solutions réalisables* (avec un test de « réalisabilité » polynomial) et une *valeur objective* calculable en temps polynomial.

3 L'approximation : une alternative pour la résolution efficace de problèmes difficiles

3.1 Le périmètre de l'approximation polynomiale

La difficulté intrinsèque d'un problème se traduit la plupart du temps par une limitation très forte de la taille des instances pouvant être résolues complètement. Dans le cas de l'optimisation auquel nous-nous restreignons, plusieurs attitudes sont alors envisageables.

Les *méthodes exactes* recouvrent l'ensemble des stratégies visant la recherche de solution(s) optimale(s). De telles méthodes ne seront évidemment pas polynomiales pour des problèmes NP-difficiles. L'objectif des recherches dans cette voie est de tenter de rendre accessible à la résolution le plus d'instances possibles, c'est à dire en règle générale, des instances de taille la plus grande qui soit. Dans le cadre de l'optimisation combinatoire, les méthodes exactes consistent essentiellement en des techniques arborescentes (de type Branch&Bound, Branch&Cut, ...) pour lesquelles on cherche à limiter au maximum l'espace de recherche par l'utilisation fine de bornes, ou des méthodes de coupe exploitant la structure du polyèdre associé à une formulation du problème comme programme linéaire en nombres entiers.

Une autre attitude possible consiste à accepter de ne pas accéder à une solution optimale (mais seulement à une « bonne » solution) au profit d'un temps de calcul meilleur. Et même, dans certains cas où les méthodes exactes sont de toute façon mises en défaut, il s'agit d'accepter une solution non optimale dans une situation où aucune solution optimale ne peut être obtenue (en tout cas garantie). On se donne alors la possibilité de rechercher un compromis acceptable entre la qualité des solutions et le temps de calcul nécessaire à leur obtention.

Il faut bien être conscient qu'une telle démarche ne peut être envisagée que pour certains types de problèmes pour lesquels une notion de « bonne solution » peut être définie. Le cadre de l'optimisation présente à ce titre l'avantage de distinguer un ensemble de solutions réalisables qui sont ordonnées en fonction de leur valeur objective. La valeur objective permet alors de représenter la qualité d'une solution au regard du problème considéré.

Les méthodes approchées recouvrent des situations très variées, tant au niveau des techniques développées qu'au niveau de leur usage et des garanties qu'elles offrent. On y distingue notamment les méthodes heuristiques (ou Méta heuristiques) et, parmi les méthodes à garanties de performance, l'approximation polynomiale. Même si cette terminologie n'est pas unanimement utilisée, elle permet de désigner deux domaines complémentaires dans leurs objectifs et leurs enjeux.

Les heuristiques correspondent de loin aux méthodes les plus employées en pratique et s'avèrent particulièrement efficaces dans certaines circonstances. Les techniques sous-jacentes sont extrêmement variées, elles marient des algorithmes combinatoires, des méthodes statistiques, des méthodes « évolutionnaires » ou encore des méthodes de recherche locale.

Elles s'inspirent parfois de paradigmes de phénomènes naturels (colonies de fourmis, recuit simulé, algorithmes génétiques, ...). Ce domaine et ses différentes branches se définissent essentiellement par une liste de méthodes avec, presque toujours en toile de fond, l'objectif de résoudre de manière effective des problèmes réels de grande taille. Les critères de validation et d'évaluation de ces méthodes sont également très variés, ils vont d'études de convergence à des évaluations par plan d'expériences sur des jeux d'instances aléatoires, sur des instances réelles ou encore sur des jeux d'instances de benchmark reconnues comme particulièrement difficiles ou en tout cas mal résolues.

L'approximation polynomiale quant à elle se définie par ses objectifs : elle limite sont champs d'investigation à des méthodes polynomiales offrant des garanties au pire cas sur la valeur des solutions obtenues. Ce domaine se définit donc au sein des méthodes approchées par un choix restrictif du compromis complexité/qualité sur lequel il s'appuie. Il s'agit en quelque sorte d'étudier les meilleures solutions qui peuvent être obtenues en temps polynomial.

La garantie au pire cas qu'un algorithme d'approximation doit satisfaire s'exprime au moyen d'un *rapport* ou *mesure d'approximation* qui compare la valeur objective de la solution approchée à la valeur optimale de l'instance et éventuellement à d'autres paramètres de l'instance. Différentes mesures peuvent être envisagées mais le schéma général reste le même :

- Un algorithme d'approximation détermine, avec une complexité polynomiale, une solution réalisable pour chaque instance.
- Pour chaque instance, la performance de l'algorithme est définie par une mesure d'approximation fonction de la valeur objective de la solution approchée et de différents autres paramètres de l'instance.
- Une analyse au pire cas permet d'établir un encadrement de la mesure d'approximation qui est valide pour toute instance. Cet encadrement définit la performance de l'algorithme pour le problème.

Ce schéma peut sembler très restrictif (il l'est), mais nous verrons que ce cadre figé par lequel l'approximation polynomiale se définit détermine ses enjeux, en particulier la possibilité de comparer les différents résultats d'approximation pour un même problème et les problèmes eux-même par rapport à leurs propriétés d'approximation. Ces enjeux sont discutés dans la section 8. Bien entendu, le cadre des méthodes approchées à garanties de performance dépasse celui de l'approximation polynomiale, le schéma ci-dessus pouvant être décliné avec d'autres types de garanties, notamment en moyenne. Pour chaque classe de garanties visée, une structuration des résultats peut être développée sur le même schéma que l'approximation, définissant ainsi un périmètre au sein des méthodes approchées.

3.2 Approximation : mesures, résultats et classes

Nous désignerons respectivement, pour une instance I, par $\lambda(I)$ et $\beta(I)$ la valeur approchée et la valeur optimale de l'instance. Lorsqu'aucune confusion n'est possible, la dépendance par rapport à I ne sera pas spécifiée. Nous utiliserons également une notion de pire valeur de

l'instance, notée $\omega(I)$ qui est définie comme la valeur optimale de l'instance obtenue en inversant le sens de l'optimisation.

Dans le cas de problèmes à objectif positif, la mesure d'approximation la plus couramment utilisée est le rapport entre la valeur approchée et la valeur optimale ; nous l'appelons *rapport standard*. Un résultat d'approximation caractérise alors la proportion de la valeur optimale qui est garantie par l'algorithme. Nous éludons ici les éventuels problèmes de définition lorsque la valeur optimale est nulle car le cœur de la problématique ne se situe pas là.

Le rapport $\lambda(I)/\beta(I)$ est compris entre 0 et 1 pour un problème de maximisation et est supérieur à 1 pour un problème de minimisation ; il est d'autant plus proche de 1 que l'instance est bien résolue. Ainsi, une garantie d'approximation correspondra à minorer le rapport d'approximation pour un problème de maximisation et à le majorer pour un problème de minimisation :

◄ **Définition** ► **3.1** ***Approximation standard.***
Soit Π *un problème de NPO,* $\mathcal{A}$ *un algorithme approché polynomial pour* Π *(*$\mathcal{A}$ *calcule une solution réalisable pour toute instance), soit* ρ *une fonction attribuant à toute instance* I *un réel* $\rho(I)$*. On dit que* $\mathcal{A}$ *garantit le rapport standard* ρ *si :*

$\forall I, \lambda(I)/\beta(I) \leq \rho(I)$ *si* Π *est un problème de minimisation,*
$\forall I, \lambda(I)/\beta(I) \geq \rho(I)$ *si* Π *est un problème de maximisation.*

Une autre mesure d'approximation peut être envisagée. Il s'agit du *rapport d'approximation différentiel* qui, pour chaque instance, compare $\lambda(I)$ à la meilleure valeur $\beta(I)$ et à la pire valeur $\omega(I)$. Le rapport associé à l'instance est $(\omega(I)-\lambda(I))/(\omega(I)-\beta(I))$; il correspond à la position de la valeur approchée entre les deux valeurs extrêmes de l'instance. On a alors une définition identique à la précédente en tenant compte du fait que le rapport se situe toujours entre 0 et 1 sans distinction du sens de l'optimisation (maximisation ou minimisation) :

◄ **Définition** ► **3.2** ***Approximation différentielle.***
Soit Π*,* $\mathcal{A}$ *et* ρ *définis comme dans la définition précédente (ici* ρ *est à valeurs entre 0 et 1). On dit que* $\mathcal{A}$ *garantit le rapport différentiel* ρ *si :*

$$\forall I, (\omega(I) - \lambda(I))/(\omega(I) - \beta(I)) \geq \rho(I)$$

Ce rapport d'approximation a été utilisé depuis longtemps pour des problèmes spécifiques ([4, 39]). Une discussion sur les mesures d'approximation à utiliser est proposée dans [16] ; en particulier le rapport différentiel y est introduit en partie par une approche axiomatique. Depuis, son emploi systématique a été étudié pour différents problèmes (cf. par exemple [14, 15, 20, 29, 30]).

À l'origine, le rapport différentiel a été proposé comme réponse à certains biais de la mesure standard qui reste la plus employée. Cette dernière crée en particulier une dissymétrie

profonde entre les problèmes de maximisation et les problèmes de minimisation qui, du point de vue de l'optimisation, peuvent être rendus équivalents par simple composition de l'objectif par une fonction décroissante. L'exemple le plus significatif concerne les problèmes de stable maximum et de couverture minimum de sommets. Un stable est un ensemble de sommets deux à deux non liés par une arête. Une couverture de sommets est un ensemble de sommets touchant toutes les arêtes ; il s'agit exactement du complémentaire d'un stable. Ces deux problèmes sont donc équivalents du point de vue de l'optimisation (maximiser la taille d'un ensemble ou minimiser celle de son complément), mais le rapport d'approximation standard ne respecte pas cette équivalence. D'ailleurs comme nous le verrons dans la partie 8, ces deux problèmes ont des comportements radicalement opposés du point de vue de l'approximation standard. Au contraire, l'approximation différentielle les rend équivalents (par axiome). Un autre biais est la sensibilité par rapport aux translations, pourtant très naturelles. Dans le cas du voyageur de commerce par exemple, il suffit d'augmenter les coûts de chaque arête d'une même constante pour rendre artificiellement le rapport standard proche de 1 alors qu'il semblerait plus naturel qu'une telle transformation n'affecte pas la difficulté du problème.

Pour chacune des deux mesures, standard ou différentielle, on différentie les résultats d'approximation en fonction de la forme de la fonction ρ. On parle notamment de rapport d'approximation constant, logarithmique, polynomial, ... Un schéma d'approximation est un cas d'approximation très favorable puisqu'il correspond à la possibilité d'obtenir tout rapport constant différent de 1 :

◄ **Définition** ► **3.3** ***Schéma d'approximation polynomiale.***
(i) *Un schéma d'approximation polynomiale (PTAS) est une suite $\mathcal{A}_p$ d'algorithmes polynomiaux garantissant le rapport (standard) $1-1/p$ pour un problème de maximisation et $1+1/p$ pour un problème de minimisation.*
(ii) *Un schéma différentiel d'approximation polynomiale (DPTAS) est une suite $\mathcal{A}_p$ d'algorithmes polynomiaux garantissant le rapport (différentiel) $1-1/p$.*
(iii) *Lorsque la complexité de $\mathcal{A}_p$ est polynomiale par rapport à n (taille de l'instance) et à p, on parle de schéma complet.*

4 Approche difficile : un problème sauvage

Afin d'ouvrir le débat sur l'approximation de **Min TSP**, nous montrons dans cette section combien il est difficile d'approcher le problème général. Plus précisément, nous montrons :

Théorème 4.1
Soit $f : \mathbb{N} \to \mathbb{N}$ une fonction numérique telle que $f(n) \leq 2^{p(n)}$ pour un polynôme p, alors si P≠NP, il n'existe pas d'algorithme polynomial garantissant le rapport $f(n)$.

La restriction sur f permet juste de garantir qu'elle est représentable en temps polynomial.

Preuve : Pour cela, nous revenons à la réduction du problème de Cycle hamiltonien. Considérons un graphe $G = (V, E)$ d'ordre $n \geq 2$, instance de **HC**. Construisons le graphe complet K_n sur E en attribuant les coûts suivants :

$$\begin{cases} c(i,j) = 1 \text{ si } (i,j) \in E \\ c(i,j) = nf(n) \text{ sinon} \end{cases}$$

Suivons exactement le même raisonnement que dans la preuve du théorème 2.1 : si G est hamiltonien, alors un tour optimal a pour valeur n et dans ce cas un algorithme à rapport $f(n)$ permettrait de construire un tour de K_n de valeur au plus $nf(n)$. Par contre, si G n'est pas hamiltonien, alors tout cycle hamiltonien de K_n a pour valeur au moins $1 + nf(n)$. Par conséquent, un tel algorithme permettrait de séparer les graphes hamiltoniens des graphes non-hamiltonien. Si P≠NP, ceci ne peut être fait en temps polynomial. ■

Ce résultat signifie que même un rapport d'approximation 2^n ne peut être garanti, ce qui correspond quasiment au pire cas envisageable pour l'approximation. À titre de comparaison, des problèmes réputés très difficiles à approcher tels que le stable maximum ou la coloration minimum admettent trivialement des algorithmes garantissant le rapport n valant la taille de l'instance.

Par contre, il ne faudrait pas en déduire qu'aucun rapport ne peut être garanti pour **Min TSP**. En fait, ce résultat signifie essentiellement qu'aucun rapport d'approximation indépendant des poids ne peut être garanti. Par contre, comme tous les tours ont le même nombre d'arêtes, il est évident que tout algorithme construisant un tour réalisable (un ordre sur les sommets) garantit le rapport d'approximation w_{max}/w_{min} où w_{max} et w_{min} désignent respectivement le poids maximum et le poids minimum des arêtes de G. En particulier, la restriction du problème au cas où les poids sont bornés par une constante admet très simplement un algorithme à rapport constant. Ceci permet d'établir que la difficulté du point de vue de l'approximation est, pour une bonne partie, attribuable aux poids. Par comparaison, le cas du problème de stable maximum et de sa version pondérée est tout à fait différent. En effet, on peut montrer que la version non pondérée (i.e. où tous les poids sont égaux) se comporte essentiellement comme la version pondérée du point de vue de l'approximation et notamment, elle n'admet pas d'algorithme à rapport constant, si P≠NP [21].

Pour ce qui concerne les rapports d'approximation dépendant des poids, la preuve du théorème précédent permet d'établir que :

Proposition 4.1 *Si P≠NP, aucun algorithme ne peut garantir le rapport* $w_{max}/(nw_{min})$.

On a donc établi un encadrement pour tout rapport d'approximation garanti par un algorithme polynomial : entre $w_{max}/(nw_{min})$ et w_{max}/w_{min}.

4.1 Min TSP général : analyse d'un algorithme glouton

Nous proposons, dans cette section, une première analyse qui permet de mettre en évidence le gain induit par une technique très simple par rapport à la borne inférieure w_{max}/w_{min}. Un *algorithme glouton* construit une solution par décisions successives (au regard d'un critère spécifique) sans retour en arrière. De tels algorithmes, très simples dans leur conception, sont fréquents en approximation et permettent souvent d'établir un premier résultat. À ce propos, il convient d'ailleurs de garder à l'esprit, pour bien appréhender ce domaine, qu'il ne suffit pas de concevoir un algorithme mais de concevoir conjointement un algorithme et son analyse. En particulier, toute amélioration sophistiquée, même quand elle s'avère très performante en pratique, ne permet pas systématiquement d'améliorer l'analyse au pire cas et donc ne présente pas nécessairement d'intérêt pour l'approximation. C'est ainsi que des algorithmes très simples, voire naïfs, mais relativement faciles à analyser permettent d'obtenir des résultats d'approximation qui ne sont parfois jamais améliorés. C'est par exemple le cas de l'algorithme glouton pour la couverture d'ensembles [10] qui garantit un rapport logarithmique par rapport à la taille du problème (ici, le nombre d'éléments à couvrir). Dans le cas de **Min TSP**, un algorithme glouton naturel, noté `Plus-Proche-Voisin`, consiste, partant d'un point, à systématiquement se diriger vers la ville non déjà visitée la plus proche.

`Plus-Proche-Voisin`
`input :` 1-Graphe complet $G = (V, E)$, arêtes valué.
`output :` Cycle hamiltonien Γ.
  Choisir un sommet initial v ;
  Tant qu'il reste des sommets non visités faire
    visiter un sommet non déjà visité le plus proche ;

Analyse de l'algorithme `Plus-Proche-Voisin`

Considérons le tour fourni par l'algorithme glouton et numérotons alors les sommets du graphe $1, \ldots, n$ dans l'ordre dans lequel ils sont visités (choisis). Pour $i \in \{1, \ldots, n\}$, on définit $i \oplus 1$ comme $i + 1$ si $i < n$ et 1 si $i = n$; de même $i \ominus 1 = i - 1$ si $i > 1$ et n si $i = 1$. Le coût de la solution gloutonne est $\lambda = \sum_{i=1}^{n} c(i, i \oplus 1)$. Considérons alors une solution optimale $(s(1), s(2), s(3), \ldots, s(n))$ où s est une permutation sur $\{1, \ldots, n\}$. Sans perte de généralité, on peut supposer $s(1) = 1$. Pour tout $i = s(j)$ considérons le sommet $s(j \oplus 1) \neq i$ situé après i dans cette solution optimale. Si $s(j \oplus 1) < i$, alors d'après le critère glouton, $c(s(j), s(j \oplus 1)) \geq c(s(j \oplus 1), s(j \oplus 1) \oplus 1)$ car sinon, l'algorithme glouton se serait dirigé vers $s(j \oplus 1) \oplus 1$ après $s(j \oplus 1)$ et non vers $s(j)$. De même si $i < s(j \oplus 1)$, alors $c(s(j), s(j \oplus 1)) \geq c(s(j), s(j) \oplus 1)$. On en déduit :

$$c(s(j), s(j \oplus 1)) \geq \min(c(s(j \oplus 1), s(j \oplus 1) \oplus 1), c(s(j), s(j) \oplus 1)) \quad (1)$$

En notant $a_j = c(s(j \oplus 1), s(j \oplus 1) \oplus 1)$ et $b_j = c(s(j), s(j) \oplus 1)$ on remarque que $\sum_{j=1}^{n} a_j = \sum_{j=1}^{n} b_j = \lambda$. On note alors $J = \{j, a_j \leq b_j\}$ et $\bar{J} = \{j, a_j > b_j\}$. Sans perte de généralité, on peut supposer $|J| \geq n/2$; on a alors :

$$\lambda \leq \sum_{j \in J} a_j + |\bar{J}| w_{max} \quad (2)$$

Par ailleurs, d'après la relation (1)

$$\beta = \sum_{j=1}^{n} c(s(j), s(j \oplus 1)) \geq \sum_{j \in J} a_j + |\bar{J}| w_{min} \quad (3)$$

Par conséquent, comme $\sum_{j \in J} a_j \geq w_{min} n/2$ et $w_{max} \geq w_{min}$, on déduit des relations (2) et (3) :

$$\frac{\lambda}{\beta} \leq \frac{w_{min} n/2 + |\bar{J}|_{max}}{n w_{min}} \leq \frac{1}{2} + \frac{w_{max}}{2 w_{min}}.$$

On en déduit que l'algorithme glouton garantit le rapport $1/2 + w_{max}/(2w_{min})$ qui vaut asymptotiquement $w_{max}/(2w_{min})$. En d'autre termes, cette borne est deux fois meilleure que la borne garantie par tout algorithme.

Nous proposons de montrer maintenant que cette analyse ne peut pas être significativement améliorée. On montre que pour tout $\varepsilon > 0$, `Plus-Proche-Voisin` ne peut garantir, pour toute instance, le rapport $(1 - \varepsilon)[1/2 + w_{max}/(2w_{min})]$.

Considérons un entier $M > 1$ et une instance de **Min TSP** constituée d'une clique (graphe complet) de taille $2k$, $k \geq 1$ telle que les arêtes $\{(1, 2), (2i, 2i \oplus 1), i = 1 \ldots k, (2j, 2j + 3), j = 1 \ldots k - 2, (2k - 2, 2k)\}$ ont pour valeur $1 = w_{min}$ et toutes les autres arêtes ont pour valeur $M = w_{max}$. Partant de 1, `Plus-Proche-Voisin` peut choisir le tour $(1, 2, \ldots n)$ de valeur $M(k - 1) + k + 1$ tandis qu'un tour optimal vaut $n = 2k$: $(1, 3, 2, \ldots, 2i, 2i + 3, 2i + 2, \ldots, 2k - 1, 2k - 2, 2k)$. Le rapport garanti pour une telle instance est donc supérieur à $([1/2 + M/2]k - 1)/k \geq (1 - \varepsilon)[1/2 + w_{max}/(2w_{min})]$ pour $k \geq 1/\varepsilon$.

Le théorème suivant résume cette analyse :

Théorème 4.2
`Plus-Proche-Voisin` garantit le rapport $1/2 + w_{max}/(2w_{min})$ *; l'analyse est asymptotiquement optimale.*

5 Δ – Min TSP : approximation à rapport constant

Le théorème 4.1 limite considérablement tout espoir de résolution efficace du problème dans le cas général, hormis des améliorations assez limitées de la borne w_{max}/w_{min} telles que celle obtenue par l'algorithme glouton. Toutefois, dans de nombreux cas, les instances intéressantes ont une structure particulière permettant d'espérer des résultats beaucoup plus optimistes. Dans cette section nous considérons le cas métrique (problème Δ – **Min TSP**) correspondant aux instances pour lesquelles les coûts sont positifs et satisfont les inégalités triangulaires. Mentionnons notamment que dans ce cas, l'analyse de `Plus-Proche-Voisin` peut être affinée pour mettre en évidence un rapport logarithmique (cf. notamment [6] (ch. 2, page 49)). Mais en fait, ce cas particulier sera l'occasion de découvrir une situation beaucoup plus favorable pour l'approximation puisqu'un rapport d'approximation constant peut être garanti. Ce type de résultat correspond au cas le plus classique en approximation.

5.1 Approcher Δ – Min TSP à partir d'un arbre couvrant minimum.

Considérons d'abord un premier algorithme très simple qui nous permettra de mettre en évidence les idées principales pour une amélioration ultérieure.

Étant donnée une clique $K_n = (V, E)$ sur n sommets, un *arbre* (couvrant) de K_n est un graphe partiel $T = (V, E'), E' \subset E$ connexe et minimal pour cette propriété (tout retrait d'une arête brise la connexité). Il s'agit de manière équivalente d'un graphe partiel connexe et sans cycle ou encore d'un graphe partiel connexe comprenant exactement $n-1$ arêtes. Le système de valuation des arêtes de G permet de définir, pour chaque arbre de G, sa valeur comme la somme des valeurs de ses arêtes. Déterminer, dans un graphe connexe G, un arbre de G de valeur minimum est un problème polynomial [25]. Un tour hamiltonien de G peut immédiatement être transformé en un arbre de G par simple retrait d'une arête ; les valeurs étant supposées positives, l'arbre associé au tour est de valeur moindre que le tour lui-même de sorte que la valeur minimum d'un tour hamiltonien est minorée par la valeur minimum d'un arbre de G.

L'idée de l'algorithme est alors de concevoir un tour hamiltonien à partir d'un arbre minimum T de G en maîtrisant l'augmentation du coût. Pour ce faire, une manière de visiter tous les sommets avec retour au point initial est d'effectuer un parcours en profondeur d'un arbre minimum. Chaque arête étant traversée deux fois par un tel parcours, la valeur totale de l'itinéraire résultant est exactement le double de la valeur de T, soit au plus le double de la valeur minimum d'un tour. Bien entendu, une telle solution ne constitue pas un tour hamiltonien puisqu'un sommet peut être visité plusieurs fois dans le parcours en profondeur. Toutefois, si les inégalités triangulaires sont satisfaites, court-circuiter les sommets déjà rencontrés dans le parcours ne peut qu'améliorer cette solution. En d'autres termes, en visitant les sommets dans l'ordre dans lequel ils sont rencontrés (pour la première fois) dans un parcours en profondeur (ordre préfixe des sommets de l'arbre), on construit une solution dont

la valeur n'excède pas le double de la valeur minimum, ce qui correspond à un algorithme approché garantissant le rapport 2.

L'algorithme de Christofides [9] est fondé sur les mêmes idées interprétées sous l'angle eulérien. Alors qu'un tour hamiltonien passe exactement une fois par chaque sommet, un tour eulérien est un cycle qui passe exactement une fois sur chaque arête. On pense immédiatement au fameux problème des ponts de Königsberg résolu par Euler (1766) ! Alors que la reconnaissance d'un graphe hamiltonien (i.e. admettant un cycle du même nom) est un problème NP-complet, le théorème d'Euler fournit un certificat d'existence d'un tour eulérien qui peut être testé en temps linéaire par rapport au nombre d'arêtes : *un graphe admet un tour eulérien si et seulement si il est connexe et n'a que des sommets de degré pair* ; on dit alors que le graphe est eulérien. En outre, la preuve (cf. par exemple [8]) fournit un algorithme polynomial (linéaire par rapport au nombre d'arêtes) pour construire un tel tour lorsqu'il existe.

L'algorithme à rapport 2 analysé précédemment peut être interprété de la manière suivante : on a construit un arbre T de valeur minimum, puis en doublant chaque arête de T, on obtient trivialement un graphe respectant le critère d'Euler. Le parcours en profondeur fournit en fait un tour eulérien du graphe ainsi obtenu qui, dans le cas d'un système de valuations respectant les inégalités triangulaires, peut immédiatement être transformé en tour hamiltonien (le graphe G est complet) sans augmentation du coût. La borne 2 provient du fait que l'on a doublé chaque arête, ce qui n'est pas nécessaire pour passer de l'arbre minimum à un graphe eulérien. On se pose donc le problème de déterminer un ensemble d'arêtes de valeur minimum à ajouter à l'arbre T pour le rendre eulérien. Pour cela, il faut ajouter au moins une arête incidente à chaque sommet de degré impair dans l'arbre T. Comme il y a un nombre pair (noté $2k$) de sommets de degré impair dans T (la somme des degrés d'un graphe est paire) et comme le graphe initial est complet, il faut au moins et il suffit d'ajouter à T, pour le rendre eulérien, k arêtes ne liant que des sommets de degré pair, c'est à dire un couplage parfait du sous-graphe de G induit par ces sommets. Pour faire cette opération en limitant au maximum l'augmentation du coût, il faut déterminer un *couplage parfait de coût minimum* dans le sous-graphe de G induit par les sommets de degré impair dans T. Ceci peut être réalisé en temps polynomial (cf. par exemple [25]).

L'algorithme de Christofides ci-après, noté `CHTFD`, résume cette démarche qui vise à améliorer l'algorithme initial.

CHRFD (Algorithme de Christofides)
input : 1-Graphe complet $G = (V, E)$, arêtes-valué ;
        les valeurs positives satisfont les inégalités triangulaires.
output : Cycle hamiltonien Γ.
(i) Construire un arbre couvrant $T = (V, H)$ de G de valeur minimum ;
$V' \leftarrow \{$ sommets de degré impair dans $T\}$;
(ii) Construire un couplage parfait M de valeur minimum sur $G' = G[V']$;
Construire un parcours eulérien du graphe $G'' = (V, H \cup M)$;
En déduire un cycle hamiltonien Γ.

Analyse de l'algorithme CHTFD

Soit T de coût $c(T)$ l'arbre construit à l'étape **(i)** et soit M de coût $c(M)$ le couplage parfait construit à l'étape **(ii)**. Pour simplifier les écritures nous associons un graphe partiel, notamment T, à un ensemble d'arêtes. Il est clair que le graphe partiel $T \cup M$ est eulérien, ce qui assure la validité de l'algorithme qui construit un cycle hamiltonien de G. Notons $\tilde{\lambda}$ la valeur de ce cycle :

$$\tilde{\lambda} = c(T) + c(M) \tag{4}$$

Par ailleurs, comme dans l'analyse initiale, un tour optimal, de valeur β, peut être transformé par suppression d'une arête en un arbre de G ; les coûts étant positifs, on a :

$$\beta \geq c(T) \tag{5}$$

Remarquons aussi que, par élimination des sommets ayant un degré pair dans T, on peut en extraire d'un tour optimal un tour ne visitant que les sommets de V'. D'après les inégalités triangulaires le coût ne peut augmenter au cours de cette transformation. Par ailleurs, ce tour sur V' est la réunion de deux couplages de G', chacun de coût au moins égal à $c(M)$. Il vient :

$$\beta \geq 2c(M) \tag{6}$$

Les inégalités triangulaires garantissent que la solution construite est de valeur $\lambda \leq \tilde{\lambda}$ et alors les expressions (4), (5) et (6) impliquent :

$$\lambda/\beta \leq 3/2 \tag{7}$$

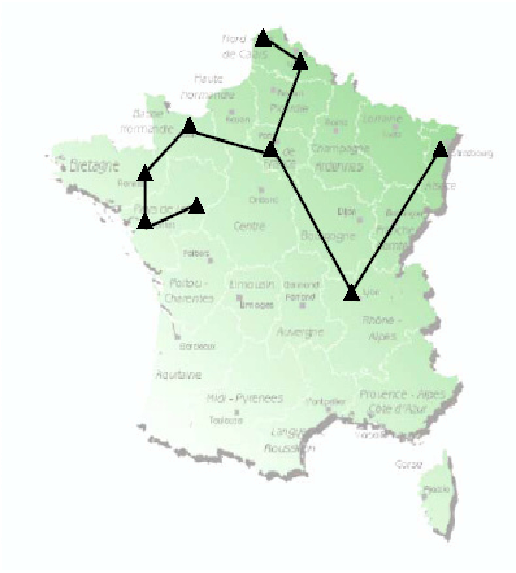
On en déduit le théorème suivant qui reste la meilleure approximation connue pour le cas métrique du voyageur de commerce :

Théorème 5.1 *[9] L'algorithme* `CHTFD` *garantit le rapport 3/2 pour le problème* $\Delta -$ **Min TSP**.

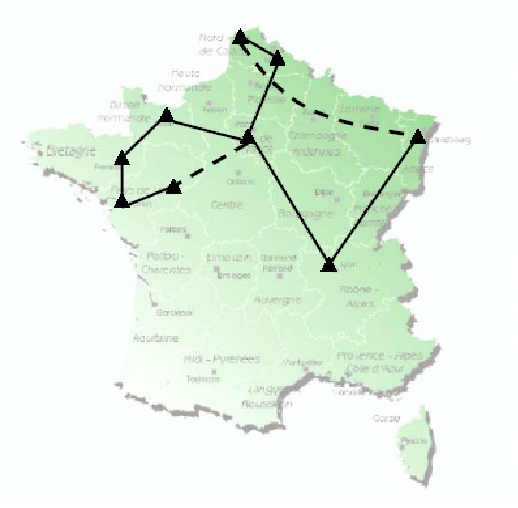
Exemple : promenade en France Imaginons l'instance de voyageur de commerce suivante correspondant à neuf villes de la moitié nord de la France. Ces données correspondent pour chaque liaison à (un) itinéraire choisi à l'avance.

	Angers	Caen	Calais	Lille	Lyon	Nantes	Paris	Rennes	Stasbourg
Angers		246	504	521	545	91	295	128	778
Caen			339	353	697	292	232	183	730
Calais				112	697	593	289	519	621
Lille					685	608	222	573	522
Lyon						634	461	770	489
Nantes							384	115	841
Paris								349	490
Rennes									831
Strasbourg									

Distances par la route entre différentes villes (km)

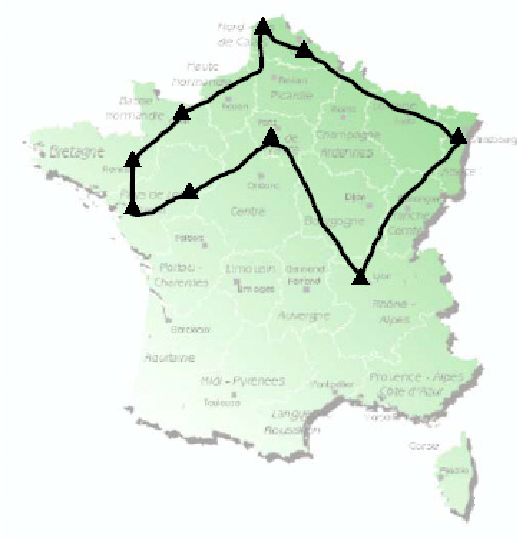


Arbre de valeur minimum (valeur 1905 km)

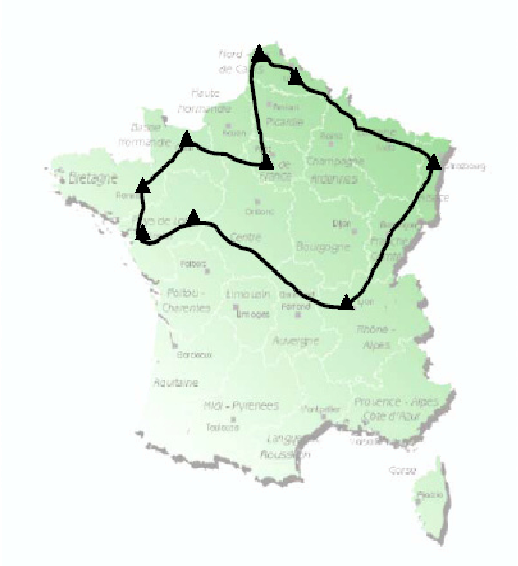


Graphe eulérien construit à partir de l'arbre (valeur 2821 km)

Les arêtes en pointillés correspondent au couplage ajouté (de valeur 916 km)



Un exemple de cycle résultant d'un parcours eulérien (valeur 2607 km)



Cycle optimal (valeur 2578 km)

5.2 Approcher Min $\text{TSP}_{1,2}$ à partir d'un 2-couplage minimum.

L'amélioration de l'algorithme de Christofides reste un problème ouvert. Le cas particulier **Min** $\textbf{TSP}_{1,2}$, pour lequel les valeurs des arêtes sont 1 ou 2, a suscité plusieurs travaux. En particulier, un algorithme à rapport 7/6 est proposé dans [34]. Le cadre de cet article ne permettrait pas d'expliquer totalement ce résultat ; néanmoins, une version très simplifiée, garantissant le rapport 4/3, permet de comprendre le principe de la démarche.

L'algorithme est fondé sur la recherche, dans un graphe complet arêtes-pondéré, d'un ensemble de cycles $M = \Gamma_i, i = 1, \dots k$ couvrant les sommets (2-couplage parfait) de valeur minimum, ce qui peut être fait en temps polynomial (cf. par exemple [31]). La valeur d'un tel 2-couplage minore celle d'un tour optimal qui correspond à un 2-couplage parfait particulier (pour lequel $k = 1$). Ce 2-couplage joue ici un rôle similaire à l'arbre dans l'algorithme précédent. La méthode consiste à assembler les cycles de M pour en faire un seul cycle en maîtrisant l'augmentation de la valeur. Cette technique de construction de tours de valeur minimum (ou maximum) est relativement classique (cf. notamment [17, 26]), nous la ré-évoquerons dans la section 7 à propos du rapport différentiel.

L'assemblage de deux cycles se fait simplement en effaçant l'arête de valeur maximum de chacun et en joignant les quatre extrémités libres pour constituer un cycle. L'algorithme assemble ainsi Γ_1 et Γ_2, puis ce nouveau cycle avec Γ_3 et ainsi de suite jusque Γ_n comme indiqué sur le schéma ci-après.

```
Assembler-Cycles1,2
input :  1-Graphe complet arêtes-valué par 1 ou 2.
output : Cycle hamiltonien Γ.
   Construire M = (Γ_1, ..., Γ_k), un 2-couplage parfait de valeur minimum.
   Γ ← Γ_1 ;
   Pour i ← 2 à k faire
         Γ ← Assembler(Γ, Γ_i)
```

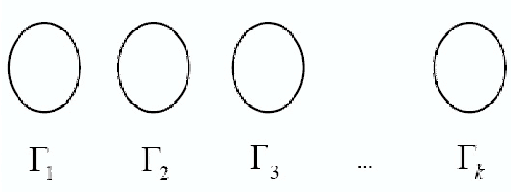
---

où **Assembler** est définie par

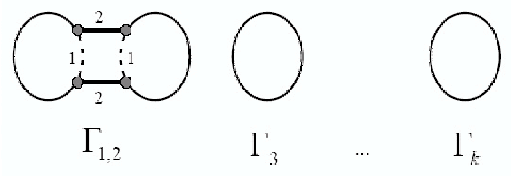
```
Assembler
input :  Deux cycles Γ, Γ'
output : Cycle.
         Effacer une arête la plus lourde de Γ et de Γ' ;
         Relier les extrémités libres pour constituer un unique cycle ;
```

Analyse de l'algorithme `Assembler-Cycles12`

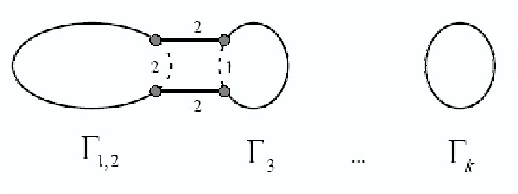
Les dessins suivants expliquent les différentes étapes de l'algorithme ainsi que les notations utilisées.



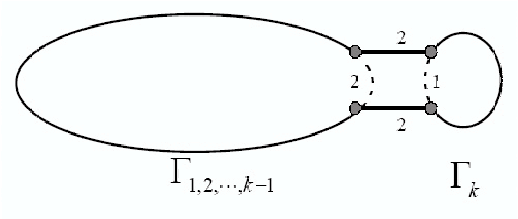
2-couplage M



Premier assemblage



Second assemblage



Dernier assemblage - Solution Γ

Soit n l'ordre du graphe G. Remarquons d'abord que chaque cycle Γ_i étant de taille au moins 3, $k \leq n/3$. Lors d'un assemblage, le surcoût peut être de 2 si deux arêtes de valeur 1 sont remplacées par des arêtes de valeur 2 et au plus 1 si l'un (au moins) des cycles à assembler contient au moins une arête de valeur 2. On note $\Gamma_{1,\ldots,p}$ l'état de Γ après $p-1$ assemblages ($p \leq k$), on montre par récurrence sur p que $\forall p \in \{1,\ldots k\}, c(\Gamma_{1,\ldots,p}) \leq \sum_{i=1}^{p} c(\Gamma_i) + p$; de plus si l'inégalité est une égalité, alors $\Gamma_{1,\ldots,p}$ contient au moins une arête de valeur 2. Pour $p = 1$, l'inégalité est stricte. Supposons maintenant la propriété vérifiée pour $p \leq k-1$ et considérons l'assemblage entre $\Gamma_{1,\ldots,p}$ et Γ_{p+1}.
Supposons d'abord que $c(\Gamma_{1,\ldots,p}) < \sum_{i=1}^{p} c(\Gamma_i) + p$. Dans ce cas :

$$c(\Gamma_{1,\ldots,p+1}) \leq c(\Gamma_{1,\ldots,p}) + c(\Gamma_{p+1}) + 2 < \sum_{i=1}^{p+1} c(\Gamma_i) + p + 2$$

et donc

$$c(\Gamma_{1,\ldots,p+1}) \leq \sum_{i=1}^{p+1} c(\Gamma_i) + (p+1)$$

L'égalité correspond nécessairement à un surcoût de 2 lors de cet assemblage, ce qui impose au moins une arête (en fait même deux arêtes) de valeur 2 dans $\Gamma_{1,\ldots,p+1}$.
Si maintenant, $c(\Gamma_{1,\ldots,p}) = \sum_{i=1}^{p} c(\Gamma_i) + p$, alors $\Gamma_{1,\ldots,p}$ contenant une arête 2 au moins, l'assemblage produit un surcoût de 1 si une arête de valeur 2 et une de valeur 1 sont remplacées par deux de valeur 2 (le nouveau cycle contient alors une arête de valeur 2) ou de 0 au plus sinon. Dans ce dernier cas l'inégalité sera stricte pour $\Gamma_{1,\ldots,p+1}$.
En appliquant cette relation pour $p = k \leq n/3$ on a : $c(\Gamma_{1,\ldots,k}) = \lambda \leq c(M) + n/3$. Comme la valeur optimale vérifie $\beta \geq c(M)$ et $\beta \geq n$, on a $\lambda \leq 4/3\beta$, d'où le résultat **(i)** suivant :

Théorème 5.2 *[34]* **(i)** `Assembler-Cycles12` *garantit le rapport 4/3 pour* **Min TSP**$_{1,2}$. **(ii)** *Le rapport 7/6 peut être garanti en temps polynomial pour* **Min TSP**$_{1,2}$.

Difficulté d'approximation

Améliorer les rapports d'approximation de $\Delta - \textbf{Min TSP}$ et **Min TSP**$_{1,2}$ reste un problème particulièrement intéressant. Parallèlement, les chercheurs du domaine mettent au point des résultats négatifs qui correspondent à des seuils d'approximation qu'on ne peut pas dépasser. De tels résultats sont très simples à obtenir pour le problème **TSP** général comme le montre le théorème 4.1.

La question de l'existence d'un schéma d'approximation polynomial pour $\Delta - \textbf{Min TSP}$ a été résolue en 1992 (par la négative) dans [3] en utilisant des techniques de preuves interactives qui ont permis, depuis un dizaine d'années, une explosion des résultats de difficulté d'approximation.

Quasiment au même moment il a été montré ([34]) que **Min TSP**$_{1,2}$ (et donc aussi $\Delta - \textbf{Min TSP}$) est APX-complet, c'est à dire complet dans la classe APX (problème admettant un algorithme à rapport constant) pour des réductions préservant les schémas d'approximation. En d'autres termes, l'existence d'un schéma d'approximation pour ce problème permettrait de construire un tel schéma pour tout problème de la classe APX et donc que P=NP. Ce résultat est montré grâce à une réduction préservant les schémas d'approximation polynomiale du problème 3-SAT connu pour être APX-complet. La démarche pour montrer de tels résultats est donc similaire à une preuve de NP-complétude en exploitant des réductions plus restrictives que les réductions de Karp (nous revenons sur ces notions en fin d'article). Ce résultat reste bien entendu valide pour le problème plus général $\Delta - \textbf{Min TSP}$; le lecteur intéressé trouvera également une autre preuve de complétude de ce problème dans [25] (page 476). Le fait qu'il n'existe pas de schéma d'approximation polynomiale signifie qu'il existe une constante qu'aucun algorithme polynomial ne peut garantir. Depuis ce résultat, plusieurs bornes explicites d'un tel seuil ont été mises en évidence, notamment si P≠NP aucun algorithme polynomial ne peut garantir le rapport $129/128 - \epsilon$ [32].

6 Le cas euclidien : schéma d'approximation polynomiale

Le cas euclidien $\ell_{\mathbf{2}} - \textbf{Min TSP}$ est très naturel : les sommets sont des points de $\mathbb{R}^d$ et chaque arête (x, y) est pondérée par $\|x - y\|_2$. En fait, de nombreuses applications s'inscrivent dans ce cas. Malheureusement le problème reste NP-difficile au sens fort [18]. Jusque récemment, l'approximation de ce cas particulier est restée relativement mystérieuse : d'une part, les résultats négatifs valides pour $\Delta - \textbf{Min TSP}$ ne semblaient pas se généraliser au cas euclidien et d'autre part aucune amélioration significative était connue par rapport au cas métrique. Juste certains résultats spécifiques à ce cas étaient connus mais sans différenciation

majeure. Nous mentionnons juste, en section 7 un algorithme de recherche locale pour lequel, du point de vue du rapport standard, le cas euclidien se démarque légèrement.

La question a été levée en 1996 par Arora ([1], intégré depuis à [2]) avec un premier PTAS (schéma d'approximation polynomiale) suivi par plusieurs améliorations [2, 28].

Théorème 6.1 *[1]* $\ell_2 -$ **Min TSP** *(cas euclidien) admet un schéma d'approximation polynomiale.*

Preuve : (pour plus de détails, cf. [2], [25] (page 578) et [38] (page 84))

Les techniques utilisées s'appliquent en fait à de nombreux problèmes géométriques. La preuve de ce résultat, très technique, ne peut être reportée dans cet article. Dans le cadre de nos objectifs nous évoquons seulement les (très) grandes lignes de la démarche d'Arora, certains de ses ingrédients étant très classiques dans l'élaboration de schémas d'approximation.

L'origine de la démarche remonte en fait à des travaux de Karp en 1977 [24] ainsi qu'à un schéma d'approximation obtenu pour le cas planaire [19]. Les trois idées essentielles sont **(1)** la définition d'instances approchées par un procédé de « scaling and rounding » de sorte qu'approcher ces instances à $(1+\epsilon)$-près suffit pour approcher le problème initial, **(2)** une discrétisation du problème limitant la recherche à un ensemble de solutions réalisables relativement réduit et contenant au moins une bonne solution du problème défini en **(1)** et enfin, la résolution par programmation dynamique du problème restreint défini en **(2)**.

On fixe a priori $\epsilon > 0$, l'objectif étant alors de concevoir un algorithme garantissant le rapport d'approximation $(1+\epsilon)$. Nous nous plaçons ici dans le cadre $\mathbb{R}^2$ qui a en fait valeur de généralité.

Phase (1)

La première phase consiste à remarquer ([25]) qu'il suffit de considérer des instances pour lesquelles :

(a) les coordonnées des points (villes) sont de la forme $1+8k, k \in \mathbb{N}$;

(b) la distance entre deux villes est au plus $\frac{64n}{\epsilon} + 16$.

En effet, étant donnée une instance de $\ell_2 -$ **Min TSP** définie par un ensemble $V \subset \mathbb{R}^2$, on note L la distance maximale entre deux points de V (villes). On définit alors une « instance approchée » V' en remplaçant chaque point $v = (v_1, v_2) \in V$ par le point $v' = (1+8\left\lfloor \frac{8n}{\epsilon L} v_1 \right\rfloor, 1+8\left\lfloor \frac{8n}{\epsilon L} v_2 \right\rfloor)$.

On se rend facilement compte, par simples manipulations de la norme que :

(i) V' satisfait les conditions **(a)** et **(b)** ;

(ii) Toute solution réalisable pour V' de valeur λ' correspond dans V à une (au moins) solution de valeur $\lambda \leq \frac{\epsilon L}{8n}(\lambda'/8 + \sqrt{2}n) \leq \frac{\epsilon L}{8n}(\lambda'/8 + 2n)$.

(iii) De même en construisant une solution pour V' à partir d'une solution pour V on a $\beta(V') \leq 8\left(\frac{8n}{\epsilon L}\beta(V) + 2n\right)$.

Notons que plusieurs points de V peuvent correspondre à un point de V' ; dans ce cas, pour recomposer une solution sur V à partir d'une solution sur V' (cas **(ii)**) tous les sommets de V se « projetant » sur un même sommet de V' sont visités consécutivement dans n'importe quel ordre. Enfin, les propriétés **(ii),(iii)** permettent d'établir (comme $\beta(V) \geq 2L$) qu'il suffit de garantir pour V' le rapport d'approximation $(1+\epsilon/2)$ pour garantir sur V le rapport $(1+\epsilon)$. Un schéma d'approximation sur V' correspond donc à un schéma d'approximation sur V. On se limite donc aux instances satisfaisant **(a)** et **(b)**.

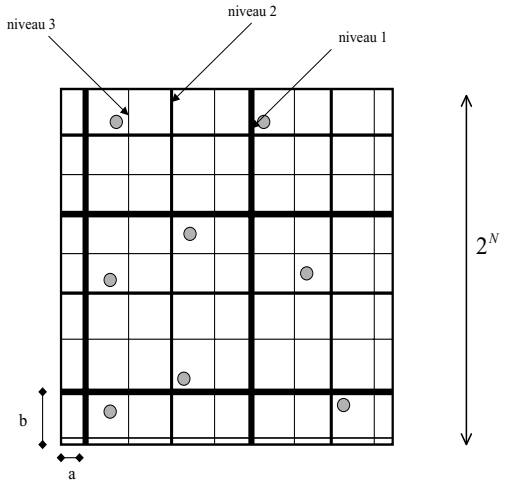
Phase (2)

Pour une instance V' satisfaisant **(a)** et **(b)**, quitte à translater le problème, on inscrit V' dans le carré $K = [0, 2^N] \times [0, 2^N]$ avec $N = \lceil log_2 L \rceil + 1$. Le carré est alors divisé récursivement en quatre, puis chaque quart en quatre et ainsi de suite pendant $N-1$ itérations. On définit ainsi un maillage de K correspondant à une grille qui passe par tous les points entiers à coordonnées paires dans K. Par hypothèse sur V', les villes à visiter sont à l'intérieur des zones ainsi définies et de plus le maillage les sépare : aucune zone ne contient plus qu'une ville.

Le processus de subdivision est organisé comme un arbre de degré 4 : la racine est K et chaque noeud correspond à un des carrés obtenus à une étape. Un carré de coté supérieur à 4 possède quatre fils dans l'arbre (ses quatre quarts). Les lignes de la grille peuvent être partitionnées en segments de différents niveaux : un segment de niveau s correspond à un coté d'un carré créé par subdivision d'un carré de niveau $s-1$. Ainsi, les segments de niveau i sont de longueur 2^{N-i}.

On fixe p un entier (choisi ultérieurement) et c une constante.

Pour chaque $i \in \{1, \ldots N-1\}$, on place lors de la division d'un carré de niveau i en quatre, un point au milieu du carré (intersection des lignes séparantes) et, sur chacun des quatre segments, on place $2^p - 1$ points répartis uniformément. On appelle ces points des « portes » que l'on va considérer comme les seuls points de passage possibles d'une zone à l'autre de la grille. Sur une ligne de niveau i, l'espace entre deux portes est donc de 2^{N-i-p}. On dira qu'un tour est *sympathique* par rapport à cette grille s'il contient tous les points de V', ne franchit les lignes verticales ou horizontales de la grille qu'en des portes et ne passe pas plus de c fois dans chaque porte. L'idée est de limiter le recherche d'une bonne solution à ce type de tours.



Grille translatée de (a, b)

Malheureusement, il se peut qu'aucun tour sympathique ne garantisse un rapport $(1+\epsilon)$. Par contre, un tel tour existe en s'autorisant une translation du maillage d'un vecteur (a, b) à coordonnées entières paires (pour garder les villes au milieu des zones). Ce résultat constitue en fait le noyau de la démonstration.

L'idée de base est la suivante : soit un couple (a, b) et la grille associée, considérons un tour optimal et transformons le pour le rendre sympathique en maîtrisant l'augmentation de sa longueur.

La transformation se fait en deux temps : **1)** à chaque fois que le tour traverse une ligne de la grille on le dévie pour passer par la porte la plus proche ; **2)** on applique ensuite une procédure permettant de réduire le nombre de passages dans chaque porte à au plus c.

On évalue alors l'augmentation de la longueur par un argument probabiliste car il est plus simple d'évaluer l'espérance de l'augmentation lorsque a et b suivent une distribution uniforme sur l'ensemble des positions possibles.

Considérons d'abord l'augmentation due à la déviation par des portes (transformation **1)**) : pour chaque déviation, le détour est au plus la distance séparant deux portes sur la ligne traversée, à savoir 2^{N-i-p} au niveau i. Par ailleurs, la probabilité que la ligne rencontrée soit de niveau i est 2^{i-N} sauf pour $i = 1$ pour lequel la probabilité est 2^{2-N} (effet de bord impliquant qu'il faut comptabiliser, dans la position originelle $(a = b = 0)$ le bord gauche et le bord bas du carré global comme lignes de niveau 1). L'espérance d'augmentation est donc $N2^{-p}$ pour chaque croisement. Par ailleurs, comme les villes ne se trouvent pas sur la grille, lors d'un parcours entre deux villes distantes de s, on traverse au plus $\sqrt{2}/2s \leq s$ fois les lignes de la grille (comparaison entre normes ℓ_1 et ℓ_2). Donc, l'espérance d'augmentation due

à la transformation **1)** ne dépasse pas $N\beta(V')/2^p$.

La transformation **2)** est un peu plus compliquée mais un calcul similaire conduit à une augmentation moyenne majorée par une expression du type $O(\beta(V')/c)$. Au total, il suffit de choisir p et c pour que l'augmentation moyenne ne dépasse pas $\epsilon\beta(V')$. Les expressions de p et c ne dépendent que de ϵ.

En fait, en augmentant p et c on peut augmenter la proportion de couples (a, b) pour lesquels il existe un tour sympathique de valeur au plus $(1+\epsilon)\beta(V')$. Ceci aurait de l'importance pour un algorithme probabiliste. Mais ici, l'analyse suffit pour conclure qu'il existe au moins un couple (a, b) pour lequel un tel tour existe.

Pour conclure la preuve, il reste à montrer que, pour chaque couple (a, b) à essayer, un tour sympathique optimal peut être trouvé en temps polynomial. Il suffira alors de couvrir tous les couples possibles et garder le meilleur tour sympathique rencontré.

Phase (3)

En fait, tous les tours sympathiques peuvent être énumérés en temps polynomial par programmation dynamique des feuilles vers la racine dans l'arbre des subdivisions. La complexité totale de l'étape de programmation dynamique est $O(n(logn)^{O(1/\epsilon)})$. Ceci devant être fait pour chaque couple (a, b), la complexité globale est $O(n^3(logn)^{O(1/\epsilon)})$. ■

La preuve est identique dans le cas de dimensions supérieures malgré une augmentation très rapide de la complexité à cause de l'énumération des vecteurs de translation de la grille. Par contre, le résultat ne se généralise pas au cas de normes l_p, $p > 2$ [6].

Au delà de PTAS ?

La question qui reste sur les lèvres après la présentation du schéma d'approximation polynomiale est de savoir si on peut encore faire mieux. Certains travaux plus récents ont pour objectif de diminuer la complexité de l'algorithme d'Arora, notamment pour éviter l'énumération des couples (a, b). Néanmoins, il convient de remarquer que toutes les complexités des schémas d'approximation pour ce problème font intervenir un terme du type $n^{1/\epsilon}$, ce qui signifie que la complexité n'est polynomiale qu'à ϵ fixé.

C'est ce point qui différentie un schéma d'approximation complet d'un schéma classique. Le premier à une complexité maîtrisée même par rapport à $1/\epsilon$. Dans ce cas, un rapport d'approximation de type $(1 + 1/P(n))$ où P est un polynôme en la taille n de l'instance, peut être garanti. La réponse à la question « qu'y a-t-il à espérer au delà d'un schéma d'approximation ? » est donc : « un schéma d'approximation complet » qui correspond, pour un problème NP-difficile, au cas le plus favorable du point de vue de l'approximation.

Du point de vue d'une résolution pratique pour un problème réel, un schéma complet correspond vraiment à une alternative intéressante à une résolution exacte éventuellement

hors d'atteinte. Malheureusement, très peu de problèmes (tel que le problème de sac à dos par exemple) admettent un tel schéma complet. En effet, différents résultats négatifs généraux ne laissent que très peu d'espace au sein de la classe FPTAS. L'un de ces résultats [6] est qu'*un problème NP-difficile à valeurs entières et polynomialement bornées ne peut admettre de schéma complet*. La justification de ce résultat consiste simplement à remarquer que dans le cas contraire un rapport du type $1/P(n)$ permettrait d'imposer entre la valeur approchée et la valeur optimale une différence inférieure à l'unité et donc nulle, ce qui ne peut évidemment pas arriver si le problème est NP-difficile et si P$\neq$NP.

Dans le cas du voyageur de commerce, ce résultat s'applique : le cas euclidien étant NP-difficile au sens fort, la restriction pour laquelle les coûts sont polynomialement bornés reste difficile. Or cette restriction vérifie les conditions du théorème et n'admet donc pas de schéma d'approximation complet.

7 Un autre point de vue : approximation différentielle

Du point de vue de l'approximation différentielle, la situation est radicalement différente de l'approximation utilisant le rapport classique. Si nous comparons déjà le problème de minimisation et celui de maximisation (les poids sont toujours supposés positifs), une première différence apparaît. En effet, du point de vue standard, les résultats très pessimistes obtenus pour la version minimisation générale (section 4) ne sont pas valables pour la version maximisation qui admet un rapport d'approximation constant ([6]). Par contre, du point de vue différentiel, ces deux problèmes sont équivalents [30, 29]. Rappelons en effet que le rapport différentiel est stable par transformation affine de l'objectif. Or il suffit de remplacer la valeur $c(u)$ de chaque arête u par $c_{max} - c(u)$ (c_{max} est la valeur maximum des arêtes) pour passer du problème de minimisation au problème de maximisation (poids positifs). Le rapport différentiel n'est pas modifié au cours d'une telle transformation. Notons par contre que l'approximation différentielle du problème **Max TSP** ne se déduit pas directement de l'approximation standard comme cela se passe souvent pour les problèmes de maximisation qui ont souvent une pire valeur nulle, ce n'est pas le cas pour le voyageur de commerce. Avant d'étudier un algorithme pour l'approximation différentielle de **Min TSP** et de **Max TSP**, remarquons une autre différence majeure entre les deux mesures approximation : dans le cas classique, nous avons mis en évidence une différence importante entre le cas général et le cas métrique. Or il suffit d'ajouter à chaque arête la valeur maximum c_{max} pour garantir les inégalités triangulaires ; le rapport différentiel étant insensible à cette transformation, il n'y a de ce point de vue aucune différence entre la version générale et la version métrique.

Cette petite introduction illustre bien combien les approches classique et différentielles peuvent s'avérer différentes, ce qui les rend complémentaires. Les problèmes **Min TSP** et **Max TSP** se prètent bien à l'approximation différentielle. Nous présentons deux algorithmes à rapport différentiel constant : le premier, décrit dans [30], exploite une méthode de recherche locale tandis que le second, issu de [29], améliore le rapport d'approximation en exploitant

une technique de 2-couplage (cf. section 5.2).

7.1 Algorithme `2-opt` et rapport différentiel

La recherche locale compte parmi les premières techniques à avoir été utilisées pour s'attaquer au problème du voyageur de commerce. Un algorithme de recherche locale est défini par une notion de voisinage de chaque solution réalisable. Partant d'une solution réalisable (fournie par exemple par un autre algorithme) on explore le voisinage de chaque solution courante pour trouver une solution strictement meilleure. Le processus s'arrête lorsqu'une solution localement optimale (i.e. optimale parmi son voisinage) est atteinte. Dans le cas du voyageur de commerce, en voyant une solution réalisable comme un ensemble d'arêtes, une notion naturelle de voisinage est de considérer, pour un k fixé, toutes les solutions pouvant se déduire d'une solution courante en ne modifiant que k arêtes, c'est à dire en remplaçant $p \leq k$ arêtes (on parle de k-échange). En d'autres termes, le voisinage d'une solution contient toutes les solutions réalisables à distance symétrique au plus $2k$ de la solution courante. L'algorithme de recherche locale exploitant ce type de voisinage est appelé k-`opt`. Il a fait l'objet de nombreux travaux (on pourra notamment se référer à [11, 27]) et constitue la base de nombreuses heuristiques avec des résultats satisfaisants.

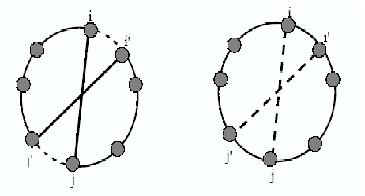
Pourtant, du point de vue de l'approximation classique, k-`opt` n'offre pas de bonnes garanties. Il reste très sensible au choix de la solution initiale de sorte que, même dans le cas de poids polynomialement bornés, aucun rapport $\alpha > 1/n$ ne peut être garanti par k-opt initialisé au hasard [6] (ch. 10 page 333) La seule borne connue est $O(\sqrt{n})$ pour le cas euclidien ([6] (ch. 10 page 335) et [11]).

Un autre problème des techniques de recherche locale pour le problème de voyageur de commerce est qu'elles ne sont pas polynomiales si on ne fait aucune hypothèse sur les valeurs : pour l'algorithme k-`opt`, chaque voisinage est de taille polynomiale mais la profondeur de recherche peut s'avérer exponentielle. C'est pourquoi, dans ce qui suit, nous nous plaçons dans le cas où les valeurs des arêtes sont bornées par un polynôme (cas polynomialement borné). Toute recherche locale ne devra alors explorer qu'un nombre polynomial de voisinages, eux-mêmes pouvant être décrits en temps polynomial. Toutefois, la complexité de la méthode mise à part, l'analyse reste valide dans le cas général.

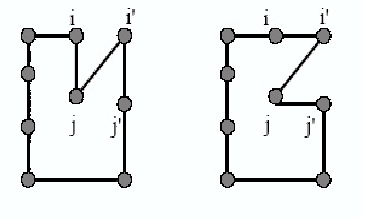
Nous présentons une analyse (cf. [30]) pour le rapport différentiel de l'algorithme `2-opt` ci-après.

2-opt
input : 1-Graphe complet arêtes-valué.
output : Cycle hamiltonien Γ.
   Initialiser Γ par une solution initiale quelconque ;
   Définir un sens de parcours sur Γ ;
   Tant qu'il existe2 arêtes (i, j) et $(i'j')$ dans Γ
         (orientées dans cet ordre par rapport au sens choisi)
         telles que $c(ij) + c(i'j') > c(ii') + c(jj')$ faire
               $\Gamma \leftarrow \Gamma \setminus \{(ij), (i'j')\} \cup \{(ii'), (jj')\}$

Dans l'algorithme, l'orientation ne sert qu'à garantir que la solution $\Gamma\setminus\{(ij), (i'j')\}\cup\{(ii'), (jj')\}$ est bien réalisable.



Exemples de 2-échanges



Analyse différentielle de 2-opt

Il est clair que l'algorithme finit puisque chaque étape améliore strictement la valeur. En outre, les hypothèses sur les pondérations garantissent que le nombre de valeurs réalisables est polynomial et donc que l'algorithme finit en temps polynomial. La solution construite est clairement réalisable et aucun 2-échange ne permet d'améliorer la valeur.

La principale différence dans le cas d'une analyse sous le rapport différentiel est la gestion de la pire valeur. En effet, ce cadre suppose non seulement l'évaluation de la valeur optimale

et de la valeur approchée ou au moins leur comparaison comme dans le cas d'une analyse classique, mais nécessite aussi l'évaluation de la pire valeur. Dans certains cas, cette dernière est très simple à évaluer. L'exemple du problème de voyageur de commerce est intéressant de ce point de vue car l'évaluation du pire est également un problème difficile (il s'agit de **Max TSP** pour le problème de minimisation **Min TSP** et réciproquement). Dans ce cas, la pire valeur doit être gérée comme la valeur optimale par la mise en évidence de bornes. L'analyse très simple qui suit illustre pleinement une démarche classique en approximation différentielle lorsqu'on ne dispose pas d'expression directe de la pire valeur. Étant données une solution optimale fixée et la solution approchée construite par l'algorithme, on met en évidence une troisième solution réalisable construite à partir des deux premières et qui permet de minorer la pire valeur.

Numérotons les sommets dans l'ordre dans lequel ils sont visités par la solution approchée Γ à partir d'un sommet choisi comme origine, une orientation de Γ étant fixée. Comme dans l'analyse de l'algorithme `Plus-Proche-Voisin` (section 4.1), on note $i \oplus 1$ (resp. $i \ominus 1$)) le successeur (resp. le prédécesseur) modulo n de i. Ainsi, pour tout sommet i, $(i, i \oplus 1)$ est une arête de Γ. Fixons de même une solution optimale Γ^* et notons s^* la permutation associée ($s^*(i)$ désigne le successeur de i sur le cycle Γ^* sur lequel a été défini un sens de parcours).

Pour chaque sommet i, on considère deux arêtes particulières de Γ : $(i, i \oplus 1)$ et $(s^*(i), s^*(i) \oplus 1)$. Un 2-échange sur Γ ne pouvant améliorer la valeur, on a :

$$c(i, i \oplus 1) + c(s^*(i), s^*(i) \oplus 1) \leq c(i, s^*(i)) + c(i \oplus 1, s^*(i) \oplus 1) \tag{8}$$

Lorsque i décrit $\{1, \ldots, n\}$, $(i, i \oplus 1)$ et $(s^*(i), s^*(i) \oplus 1)$ décrivent les arêtes de Γ. De même $(i, s^*(i))$ décrit les arêtes de Γ^*.

Remarquons par ailleurs que l'ensemble des arêtes $(i \oplus 1, s^*(i) \oplus 1)$ est un cycle hamiltonien Γ'. On en déduit donc, en sommant la relation (8) pour tout i :

$$2\lambda(I) = 2c(\Gamma) \leq c(\Gamma^*) + c(\Gamma') \leq \beta(I) + \omega(I) \tag{9}$$

Le rapport différentiel étant croissant en ω pour un problème de minimisation, on en déduit :

$$\frac{\omega(I) - \lambda(I)}{\omega(I) - \beta(I)} \geq \frac{2\lambda(I) - \beta(I) - \lambda(I)}{2\lambda(I) - \beta(I) - \beta(I)} = \frac{1}{2} \tag{10}$$

d'où le résultat suivant :

Théorème 7.1 *[30]*
`2-opt` *garantit le rapport différentiel* $1/2$. *Tout sous-problème de* **Min TSP** *pour lequel* `2-opt` *finit en temps polynomial est* $1/2$*-approché du point de vue différentiel.*

Dans l'article [30], différents cas garantissant une complexité polynomiale pour l'algorithme `2-opt` sont discutés. Toutefois, le résultat que nous présentons maintenant ([29]) améliore le rapport 1/2 et n'a aucune restriction quant à la forme des pondérations.

7.2 2-couplage et rapport différentiel

Le second algorithme reprend la technique de 2-couplage utilisée pour l'approximation classique de **Min TSP**$_{1,2}$ (section 5.2) en l'adaptant pour une analyse différentielle. Nous explicitons l'algorithme et son analyse dans le cas où le nombre de cycles du 2-couplage est pair, le cas impair est similaire dans la construction et dans la preuve.

`Assembler-Cycles (Cas` k `pair)`
`input :  Graphe complet arêtes valué.`
`output : Cycle hamiltonien` Γ.
`Construire` $M = (\Gamma_1, \ldots, \Gamma_k)$`, un 2-couplage parfait de valeur minimum ;`
`Si` $k = 1$ `Alors` $T \leftarrow M$
`Sinon Si` k `est pair Alors`
`Pour tout` $i \neq k$ `sélectionner` $x_1^i, x_2^i, x_3^i, x_4^i$ `consécutifs sur` Γ_i ;
`Sélectionner` $x_0^k, x_1^k, x_2^k, x_3^k$ `consécutifs sur` Γ_k ;
$S_1 \leftarrow \{(x_1^1, x_2^1), \ldots (x_1^{k-1}, x_2^{k-1}), (x_0^k, x_1^k)\}$;
$E_1 \leftarrow \{(x_1^1, x_1^2), (x_2^2, x_2^3), \ldots, (x_1^{2i+1}, x_1^{2i+2}), (x_2^{2i+2}, x_2^{2i+3}), \ldots, (x_1^{k-1}, x_1^k), (x_0^k, x_2^1)\}$;
$T_1 \leftarrow ((\Gamma_1 \cup \Gamma_2, \ldots \cup \Gamma_k) \setminus S_1) \cup E_1$;

$S_2 \leftarrow \{(x_2^1, x_3^1), \ldots (x_2^{k-1}, x_3^{k-1}), (x_1^k, x_2^k)\}$;
$E_2 \leftarrow \{(x_2^1, x_2^2), (x_3^2, x_3^3), \ldots, (x_2^{2i+1}, x_2^{2i+2}), (x_3^{2i+2}, x_3^{2i+3}), \ldots, (x_2^{k-1}, x_2^k), (x_1^k, x_3^1)\}$;
$T_2 \leftarrow ((\Gamma_1 \cup \Gamma_2, \ldots \cup \Gamma_k) \setminus S_2) \cup E_2$;

$S_3 \leftarrow \{(x_3^1, x_4^1), \ldots (x_3^{k-1}, x_4^{k-1}), (x_2^k, x_3^k)\}$;
$E_3 \leftarrow \{(x_3^1, x_3^2), (x_4^2, x_4^3), \ldots, (x_3^{2i+1}, x_3^{2i+2}), (x_4^{2i+2}, x_4^{2i+3}), \ldots, (x_3^{k-1}, x_3^k), (x_2^k, x_4^1)\}$;
$T_3 \leftarrow ((\Gamma_1 \cup \Gamma_2, \ldots \cup \Gamma_k) \setminus S_3) \cup E_3$;

`Si` k `est impair Alors ...`

$\Gamma \leftarrow argmin[c(T_1), c(T_2), c(T_3)]$

Analyse différentielle de l'algorithme `Assembler-Cycles`

À partir du 2-couplage M, l'algorithme construit trois solutions réalisables (tours) notées T_1, T_2 et T_3 et choisit ensuite pour Γ la meilleure. Comme le montre l'analyse ci-après, cette construction permet non seulement de déterminer la solution Γ, mais aussi, grâce aux « mauvaises parties » des solutions T_1, T_2 et T_3, d'exhiber une quatrième solution Γ' suffisamment éloignée de Γ. Il s'agit donc de déterminer conjointement deux solutions pour le problème, une « bonne » et l'autre « mauvaise ».

Notons $M = \cup_{i=1}^{k} \Gamma_i$ le 2-couplage de valeur $c(M)$. Chacun des trois tours T_i construits par l'algorithme est un 2-couplage parfait particulier, donc est de coût au moins $c(M)$ (valeur d'un couplage minimum). On note D_i la différence $D_i = c(T_i) - c(M) \geq 0$ qui correspond à ce qu'on perd pour passer de M à une solution réalisable (assemblage des cycles). On a alors :

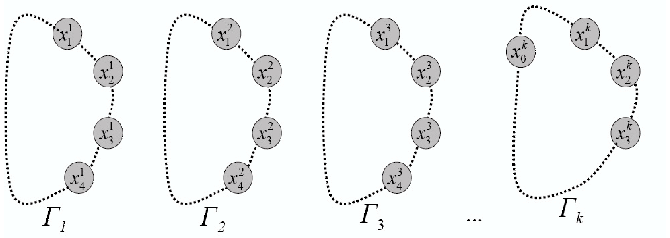
$$c(\Gamma) \leq \frac{1}{3}(c(T_1) + c(T_2) + c(T_3)) = c(M) + \frac{1}{3}(D_1 + D_2 + D_3) \tag{11}$$

Venons en maintenant à l'évaluation (minoration) de la pire valeur. Pour cela on remarque que :

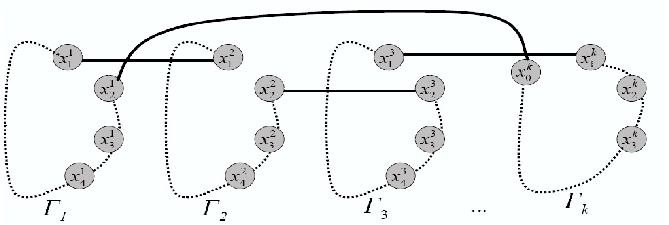
$$\Gamma' = [M \setminus (S_1 \cup S_2 \cup S_3)] \cup E_1 \cup E_2 \cup E_3$$

est un tour réalisable (cf. figures ci-dessous). On en déduit :

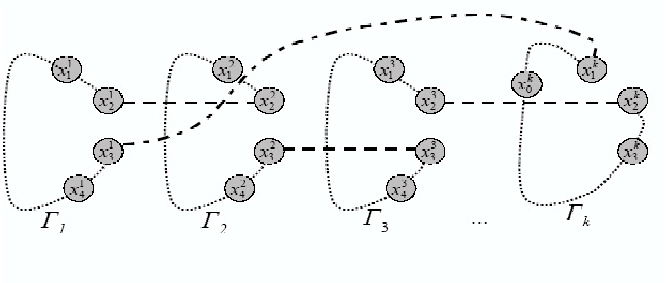
$$\omega(I) \geq c(\Gamma') = c(M) + (D_1 + D_2 + D_3) \tag{12}$$



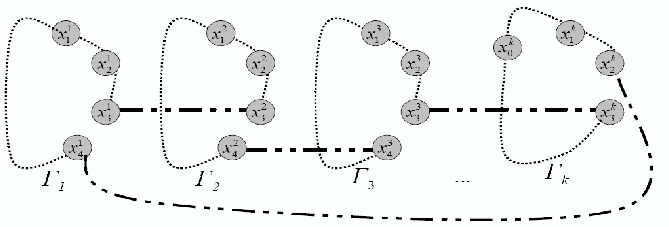
Le 2-couplage parfait M pour $k = 4$



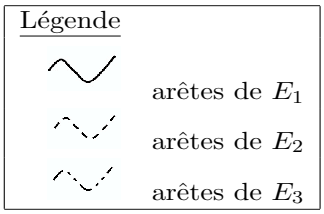
Le tour T_1 pour $k = 4$

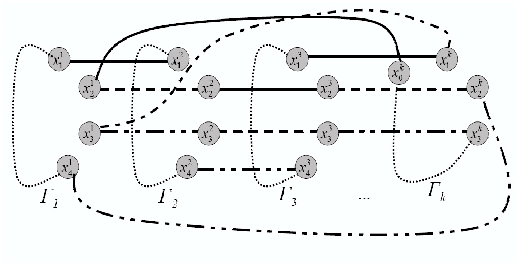


Le tour T_2 pour $k = 4$



Le tour T_3 pour $k = 4$





Le tour Γ' pour $k = 4$

Par ailleurs, M étant une solution optimale, on a :

$$\beta(I) \geq c(M) \tag{13}$$

En combinant les relations (11), (12) et (13), on obtient, pour I instance de **Min TSP**,

$$c(\Gamma) \leq \frac{1}{3}\omega(I) + \frac{2}{3}\beta(I) \Leftrightarrow \frac{c(\Gamma) - \omega(I)}{\beta(I) - \omega(I)} \geq \frac{2}{3}$$

Cette analyse permet donc d'établir le résultat suivant :

Théorème 7.2 *[29]*
L'algorithme `Assembler-Cycles` *garanti le rapport différentiel 2/3 pour* **Min TSP** *; par équivalence,* **Max TSP** *admet une 2/3-approximation du point de vue du rapport différentiel.*

En outre, l'analyse est optimale en ce sens qu'on peut exhiber des instances pour lesquelles ce rapport est atteint. Dans l'article [29], il est également prouvé que pour les problèmes **Min TSP**$_{1,2}$ et **Max TSP**$_{1,2}$, le rapport 3/4 peut être garanti par un algorithme du même type. De plus, ces deux problèmes et donc a fortiori **Min TSP** et **Max TSP** n'admettent pas de schéma d'approximation polynomiale pour le rapport différentiel ; des bornes explicites sont même proposées.

8 Conclusion : méthodes et enjeux de l'approximation

Conformément aux objectifs que nous nous étions fixés, la sélection de résultats sur l'approximation de **Min TSP** permet de se faire une première idée de ce domaine. Bien entendu, nous n'en avons parcouru qu'un champs très restreint. Toutefois, ces exemples illustrent assez bien les différents types de résultats d'approximation que l'on peut escompter ainsi que les techniques les plus simples permettant d'y mener. En guise de conclusion, nous pouvons tout d'abord dégager quelques idées générales sur les techniques et les niveaux d'approximation les plus courants. Puis, dans un second temps, ceci nous permettra d'évoquer les principaux enjeux du domaine ainsi que la face immergée de l'iceberg.

8.1 Les types d'algorithmes : quelques grands classiques

Du point de vue des méthodes, ce tour d'horizon fournit des exemples de différentes techniques algorithmiques très courantes en approximation, notamment :

- **Algorithme glouton** : de très nombreux résultats d'approximation reposent sur ces méthodes simples qui allient souvent pertinence et facilité d'analyse. Sans doute les premiers algorithmes à regarder face à un nouveau problème.

- **Recherche locale** : largement utilisées en approximation et pour des heuristiques. La classe GLO regroupe les problèmes pour lesquels un algorithme de recherche locale permet de garantir une approximation constante. Une classe similaire est définie pour l'approximation différentielle (D-GLO) (cf. par exemple [5]), l'analyse de `2-OPT` indique que **TSP** est dans cette classe.

- Utilisation d'un **problème relaxé** : ce type d'approche est très répandu, par exemple pour la programmation linéaire en nombres entiers ; l'idée est de trouver un problème relaxé (on ajoute des solutions réalisables en retirant des contraintes) plus facile à résoudre. Une solution relaxée risque de ne pas être réalisable pour le problème d'origine mais peut parfois être exploitée pour trouver une (bonne) solution du problème initial. Par ailleurs la valeur optimale du relaxé, lorsqu'on peut la déterminer, fournit une borne (inférieur dans le cas de la minimisation) pour le problème initial, les méthodes de recherche arborescente utilisent très souvent cette propriété. Dans le cadre combinatoire pour **TSP**, l'algorithme de Christofides ou les algorithmes à base de 2-couplage peuvent être vus comme des techniques de relaxation. Le problème relaxé est ici combinatoire : l'arbre minimum pour l'algorithme de Christofides et problème de 2-couplage dans l'autre exemple. Il s'agit dans ce cas de relaxations polynomiales, la technique consiste alors à déterminer une solution relaxée et la transformer en solution réalisable en maîtrisant la détérioration de la valeur. Dans le cadre de l'approximation, l'intérêt d'une telle démarche est de pouvoir exploiter pour l'analyse au pire cas, la valeur optimale du relaxé comme borne de la valeur optimale du problème initial. Les deux exemples rencontrés l'illustrent bien. Pour n'en citer qu'un autre, le plus que célèbre algorithme de couplage pour la couverture de sommets (rapport d'approximation 2) ([18], page 134) est de ce type.

- Enfin, dans le cas du schéma d'approximation polynomiale, les techniques d'**arrondi** et de **programmation dynamique** sont très courantes. L'exemple classique du schéma d'approximation pour le problème de sac à dos est notamment conçu sur la même démarche : définition d'un problème approché et résolution de ce dernier par programmation dynamique (cf. par exemple [38], page 69 ou pour plus de détails [6] page 69).

Bien entendu, les techniques classiques d'approximation ne s'arrêtent pas là. Mentionnons notamment les techniques à base de programmation mathématique, en particulier linéaire

ou quadratique qui sont à classer au registre des méthodes de relaxation et qui donnent d'excellents résultats pour certains problèmes (notamment pour le problème de stabilité dans les graphes). Le lecteur intéressé trouvera dans [22] ou encore dans [6] (ch. 2) des exemples variés des principales méthodes, notamment pour celles que nous n'avons pas abordées.

8.2 Les classes d'approximation : structurer NPO

Les résultats que nous avons mentionnés couvrent à peu près toutes les situations que l'on peut rencontrer. Nous avons vu des résultats négatifs, déterminant une limite aux possibilités d'approximation d'un problème, et des résultats positifs correspondant à la donnée d'un algorithme et de son analyse au pire cas. Les types d'approximation que nous avons rencontrés sont très variés : approximation à rapport constant, schéma d'approximation et approximations dépendant de l'instance. L'ensemble de ces résultats pouvant être décliné pour l'un ou l'autre des rapports d'approximation évoqués et il n'est pas exclu de définir de nouvelles mesures.

La diversité des résultats d'approximation et la possibilité de les différencier par des résultats négatifs expliquent en grande partie l'intérêt pour ce domaine et détermine son enjeu principal. En effet, souvenons nous de la définition de la classe NP-complet [18] : il s'agit d'une classe d'équivalence pour les réductions polynomiales. Pour les problèmes d'optimisation dont la version décision est NP-complète, ceci signifie que tous ces problèmes sont polynomialement équivalents tant que l'on s'intéresse à leur résolution exacte.

Par contre, du point de vue de l'approximation, ils s'avèrent non équivalents. Ainsi, l'approximation polynomiale permet-elle de distinguer une structure au sein de la classe NPO (problèmes d'optimisation dont la version décisionnelle est NP) et, par voie de conséquence, une structure au sein de NP, au moins pour l'ensemble des versions décisionnelles de problèmes d'optimisation. Le fait d'utiliser en parallèle deux rapports d'approximation met même à notre disposition une structure bidimensionnelle de NPO. Les nombreux résultats d'approximation différentielle (cf. notamment [14, 15, 20, 29, 30]) ont permis de mettre en évidence des situations très variées dont les quelques exemples décrits ici ne peuvent rendre compte. Ces derniers montrent néanmoins un exemple de problème qui peut être approché par un rapport constant dans le cadre différentiel et pas dans le cadre classique. Des situations symétriques ont également été mises en évidence (ainsi la couverture de sommets, 2-approchée dans le cadre classique, n'admet pas d'approximation constante dans le cadre différentiel). Dans d'autres cas, les comportements du point de vue de chaque rapport sont comparables de sorte que les résultats actuellement connus permettent de décrire de nombreuses combinaisons possibles entre l'approximation classique et l'approximation différentielle d'un même problème. Cette gamme de résultats valide a posteriori chacune des deux approches qui apparaissent comme complémentaires et chacune porteuse d'information. Ainsi, NPO peut être décomposé, pour chaque rapport d'approximation en différents niveaux d'approximation, le préfixe « D » faisant allusion au cadre différentiel :

APX (DAPX) : La classe des problèmes admettant un algorithme à rapport (resp. rapport différentiel) constant.

PTAS (DPTAS) : La classe des problèmes admettant un schéma d'approximation polynomiale (resp. schéma différentiel).

FPTAS (FDPTAS) : La classe des problèmes admettant un schéma d'approximation polynomiale complet (resp. schéma différentiel complet).

Log-APX (Log-DAPX) : La classe des problèmes admettant un algorithme à rapport (resp. rapport différentiel) logarithmiquement borné. En fait, la question de trouver un problème naturel dans la classe Log-DAPX reste ouvert.

Poly-APX (Poly-DAPX) : La classe des problèmes admettant un algorithme à rapport (resp. rapport différentiel) polynomialement borné.

Exp-APX (Exp-DAPX) : La classe des problèmes admettant un algorithme à rapport (resp. rapport différentiel) exponentiel.

En fait, cette classification peut être raffinée à volonté en considérant des classes de fonctions intermédiaires. Dans le cadre différentiel, la classe 0-DAPX a été mise en évidence comme contenant des problèmes naturels. Il s'agit de problèmes pour lesquels aucun rapport d'approximation différentielle ne peut être établi ; bref ... des monstres parmi lesquels pourtant des problèmes d'apparence bénigne [5] !

Nous avons représenté ci-après la structure de NPO suivant les deux rapports d'approximation. Nous y avons placé, en gras, les problèmes dont il a été question dans cet article, et en non-gras quelques exemples très classiques d'autres problèmes. Nous rappelons rapidement leur définition intuitive (pour de plus amples détails, cf. [6]) :

Max Matching : déterminer dans un graphe un couplage (ensemble d'arêtes deux à deux non adjacentes) de cardinal maximum. Ce problème est polynomial (cf. par exemple [25]).

Max Independent set : déterminer dans un graphe un stable (ensemble de sommets deux à deux non liés par une arête) de cardinal maximum.

Min Coloring : colorier les sommets d'un graphe avec un nombre minimum de couleurs de sorte que deux sommets adjacents ne soient pas de la même couleur (partition en stables).

Min Bin Packing : ranger des objets de taille inférieure à 1 dans un minimum de boites, chacune de capacité 1.

Min Set Cover : étant donné un système d'ensembles couvrant un ensemble de base E, déterminer un sous-système de cardinal minimum couvrant E.

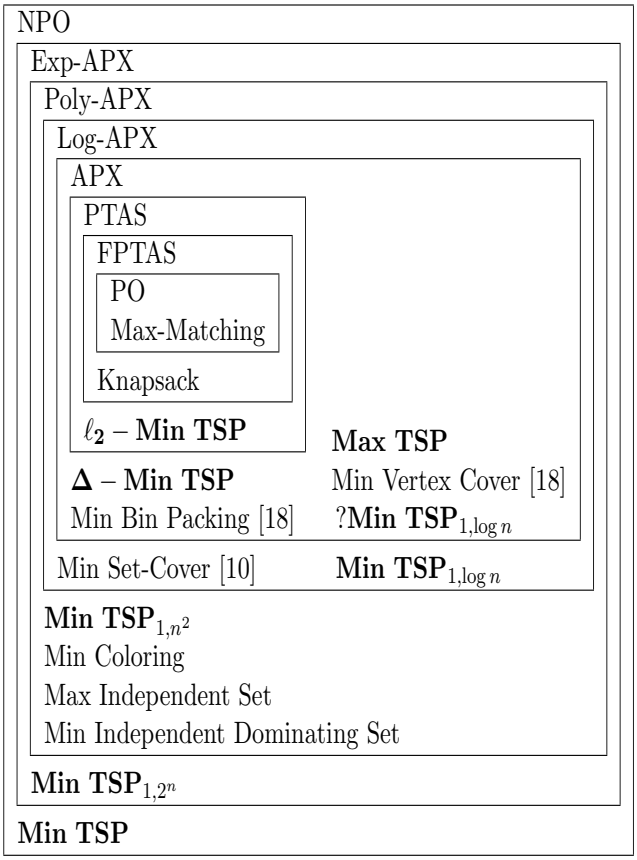
Min Vertex Cover : déterminer, dans un graphe, une couverture de sommets (ensemble de sommets touchant toutes les arêtes, i.e. le complémentaire d'un stable) de cardinal minimum.

Min/Max Knapsack : programmation linéaire à une contrainte.

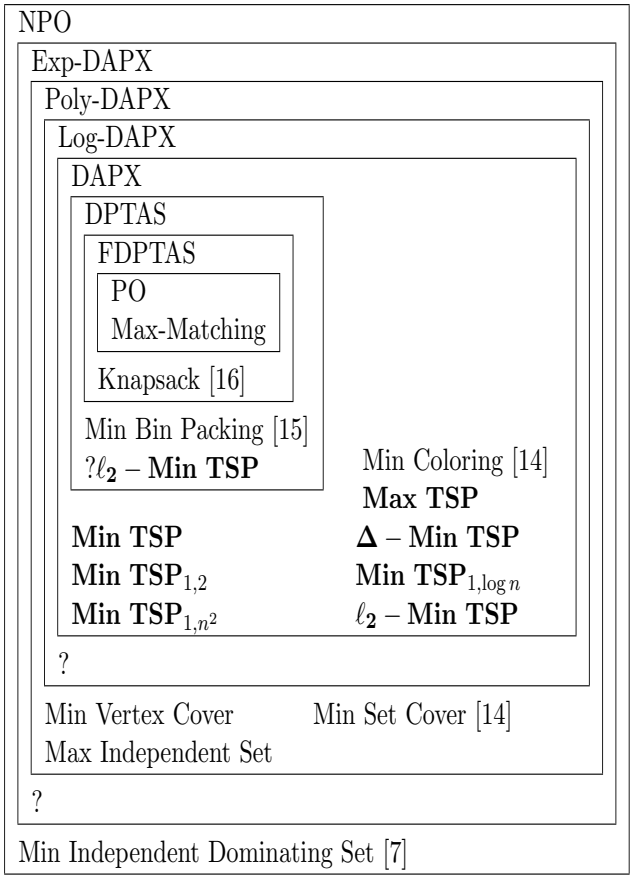
Min Independent Dominating Set : déterminer dans un graphe un stable maximal (pour l'inclusion) de cardinal minimum.

Les « ? » correspondent à des questions, non nécessairement difficiles mais pour lesquelles il n'y a pas encore, à ma connaissance, de réponse satisfaisante (notamment la connaissance de problèmes naturels pour les classes qui apparaissent vides sur les schémas ci-après).

Enfin, vous remarquerez des problèmes étranges dans ce schéma, à savoir **Min TSP**$_{1,\log n}$, **Min TSP**$_{1,n^2}$ et **Min TSP**$_{1,2^n}$, les versions de **Min TSP** pour lesquelles les poids ne peuvent prendre respectivement que les valeurs 1 ou $\log n$, 1 ou n^2 et 1 ou 2^n, n étant l'ordre du graphe. Ces problèmes ne sont évidemment pas naturels. Leur rôle dans la hiérarchie est de montrer combien, dans le cas du rapport standard, les résultats dépendent des poids alors que ceci n'est pas vrai dans le cas du rapport différentiel. Par contre, étudier l'approximation différentielle de ℓ_2 – **Min TSP** reste une question intéressante.



Structure de NPO (rapport d'approximation standard)



Structure de NPO (rapport d'approximation différentiel)

8.3 Réductions en approximation

Structurer NPO est sans doute l'un des principaux enjeux de l'approximation. Ce domaine s'inscrit ainsi dans le prolongement direct de la théorie de la complexité. On comprend mieux alors les règles que ce domaine s'impose et qui peuvent sembler trop restrictives dans le cadre d'une recherche de bonnes solutions. Pourquoi se restreindre au cadre polynomial ? Pourquoi imposer des garanties au pire cas qui, souvent s'avèrent très pessimistes par rapport au comportement réel des algorithmes ? Une réponse est que, sans cadre strict on ne peut espérer de résultats structurels, notamment des résultats négatifs (de difficulté) qui permettent de séparer des classes. Ainsi, la question devient : « pourquoi ne pas choisir d'autres règles ? » Tout à fait ! L'approximation est un exemple de compromis complexité/qualité directement inspiré de la théorie de la complexité mais d'autres compromis peuvent (et parfois sont) étudiés. Autant de travaux complémentaires au cadre que nous venons de voir. Pour n'en citer qu'un exemple, certains travaux ne s'intéressent qu'au cadre des complexités très faibles, notamment linéaires. La séparation entre polynomial et non-polynomial devient alors linéaire / non linéaire [13]. Se pose notamment la question de l'approximation en temps linéaire de problèmes polynomiaux.

Dans la logique de la théorie de la complexité se pose la question de **réductions en approximation**. En effet, la structure de NPO du point de vue de l'approximation montre que les réductions polynomiales [23] ne préservent pas les propriétés d'approximation. Il s'agit donc de définir des réductions plus restrictives qui permettent de transférer d'un problème à un autre des résultats d'approximation. Nous avons implicitement rencontré de telles réductions. Dans le cadre classique, les résultats de difficulté d'approximation de **TSP** sont obtenus par réduction. De même, nous avons évoqué l'équivalence, du point de vue du rapport différentiel, des versions maximisation et minimisation de **TSP**, là encore il s'agit de réductions. L'idée est toujours la même : à partir d'une instance d'un problème **A** dont on cherche une approximation, on construit une (ou plusieurs) instance(s) d'un problème **B** dont l'approximation a déjà été étudiée de telle sorte que toute solution approchée de l'instance (ou des instances) de **B** peut (peuvent) être convertie(s) en une solution approchée pour l'instance de **A** en maîtrisant l'évolution du rapport d'approximation. Tout résultat d'approximation de **B** peut alors se transférer (éventuellement sous une autre forme) en un résultat sur **A** et de même tout résultat de difficulté d'approximation sur **A** donne lieu à un résultat de difficulté d'approximation pour **B**. On dit ici que ***A*** *se réduit (en approximation) à* ***B***. Le lecteur intéressé par de telles transformations peut par exemple consulter [6] (ch. 8). Dans le cadre différentiel, de telles réductions existent [14]. Certaines même permettent de lier les deux cadres, classique et différentiel. Ainsi, par exemple, dans [15], une réduction de ce type nous permet d'obtenir un schéma d'approximation différentiel pour le Bin Packing à partir de l'approximation classique de ce même problème. En fait, toutes les combinaisons sont intéressantes à regarder. De nombreux résultats d'approximation actuels sont obtenus par de tels outils.

Enfin, à l'image des résultats de NP-complétude, les réductions en approximation donnent

lieu à des résultats de complétude au sein des classes d'approximation. Ainsi par exemple, le résultat d'APX-complétude que nous avons mentionné pour **Min TSP**$_{1,2}$ signifie que tout problème de la classe APX se réduit à **Min TSP**$_{1,2}$ par une réduction préservant les schéma d'approximation. Ainsi, un schéma pour **Min TSP**$_{1,2}$ serait immédiatement transformé en un schéma pour tout problème d'APX. Du point de vue structurel, la complétude permet, au sein d'une classe (APX dans notre exemple) de distinguer ce que l'on peut considérer comme les problèmes les plus difficiles (au sens de l'approximation) de la classe. Le lecteur intéressé pourra se référer, par exemple à [6] (ch. 8) pour le cadre classique et à [5] pour de premiers résultats de complétude dans le cadre différentiel.

8.4 Enjeux

À l'issue de ce bref aperçu du domaine des algorithmes d'approximation polynomiale, je souhaiterais conclure en rappelant les deux principaux enjeux, selon moi, de l'approximation et faire quelques remarques sur les différents types de résultats attendus dans le domaine.

Le premier enjeu est, bien entendu, la possibilité de résoudre des problèmes difficiles par des algorithmes à la fois polynomiaux et offrant des garanties absolues (valables pour toute instance) d'approximation. À ce titre, il faut voir ce domaine, au sein de la recherche opérationnelle, comme complémentaire des techniques de résolution exacte et des techniques heuristiques.

Un second enjeu s'inscrit plus dans la continuité de la théorie de la complexité : structurer NPO (pour mieux la comprendre) en classes d'approximation correspondant à différents niveaux de difficulté des problèmes, lier ces classes par des réductions permettant de les comparer et les compléter par des classes de complétude.

Mentionnons ici tout un pan du domaine que nous n'avons pas évoqué. Dans cette démarche de structurer, une approche duale des classes d'approximation consiste à différencier les problèmes de manière syntaxique à partir d'une écriture en termes de logique des problèmes d'optimisation (on parle alors de classes syntaxiques) et d'étudier les propriétés d'approximation des différentes classes. Cette voie, initiée notamment dans [33] a joué entre autres un rôle important dans la découverte de résultats de complétude.

Du point de vue des résultats, nous en avons mis en évidence de trois types. Les *résultats* positifs correspondent à la mise au point d'un algorithme et de son analyse. Nous avons déjà insisté sur différentes techniques utilisées dans ce cadre ; ajoutons-y désormais les réductions en approximation qui sont à la base de nombreux résultats. Du point de vue des classes d'approximation, un résultat positif correspond à l'appartenance à une classe.

Les *résultats négatifs* correspondent à une impossibilité, sous une hypothèse de théorie de la complexité (du type P≠NP), d'obtenir tel résultat d'approximation pour un problème. Du point de vue structurel il s'agit donc d'exclure un problème d'une classe et par conséquent de différencier deux classes. C'est la conjonction de résultats positifs et négatifs qui permet

d'établir la hiérarchie de NPO en classes d'approximation. Du point de vue des techniques, les réductions en approximation jouent un rôle central pour l'élaboration de tels résultats.

Mentionnons ici un autre pan du domaine totalement élidé dans cet article : les *systèmes de preuves interactives* (PCP) qui ont joué un rôle majeur dans l'obtention des résultats négatifs actuels. Pour simplifier de manière outrancière, un tel système peut être vu comme une généralisation des machines de Turing non déterministe conduisant (« PCP-theorem » [3]) à une caractérisation probabiliste de NP. L'utilisation de tels systèmes dans le cadre de l'approximation et cette caractérisation ont ouvert la voie à de très nombreux résultats négatifs spectaculaires. Le lecteur pourra se référer par exemple à [6] (ch. 6-7) et à [22] (ch. 10).

Enfin, les *résultats conditionnels* lient l'approximation de différents problèmes. Il s'agit, de manière plus ou moins directe, de réductions en approximation. Nous avons déjà eu l'occasion d'insister sur leur rôle pour l'obtention de nouveaux résultats, qu'ils soient positifs ou négatifs. Du point de vue structurel, il s'agit de lier différentes classes d'approximation. Elles donnent également accès à des résultats de complétude permettant ainsi de définir de nouvelles classes, le tour (non hamiltonien) est bouclé.

Remerciements

Cet article a été réalisé pendant ma visite à l'Ecole Polytechnique de Lausanne, au sein de la chaire de Recherche Opérationnelle (ROSE). Je tiens tout particulièrement à remercier toute l'équipe pour son accueil. Merci également à Éric Soutif pour son aide (à distance) pendant la réalisation de ce travail.

Références

[1] S. Arora, *Polynomial-time Approximation Schemes for Euclidean TSP and other Geometric Problems*, Proc. of the 37th Annual IEEE Symposium on Foundations of Computer Science, 2-12, 1996.

[2] S. Arora, *Polynomial-time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems*, Journal of the ACM **45**(5) 753-782, 1998.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, *Proof verification and hardness of approximation problems*, Proc. of the 33rd Annual IEEE Symposium on Foundations of Computer Science, 14-23, 1992.

[4] G. Ausiello, A. D'Atri and M. Protasi, *Structure preserving reductions among convex optimization problems*, Journal of Computer and System Sciences **21**, 136-153, 1980.

[5] G. Ausiello, C. Bazgan, M. Demange and V. Th. Paschos, *Completeness in differential approximation classes*, Proc. of 28th International Symposium on Mathematical Foundations of Computer Science (MFCS), ed. B. Roban and P. Vojtás, LNCS 2747, 179-188, 2003.

[6] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela and M. Protasi, *Complexity and approximation (combinatorial optimization problems ans their approximability properties)*, Springer-Verlag, 1999.

[7] C. Bazgan and V. Th. Paschos, *Differential approximation for optimal satisfiability and related problems*, European Journal of Operational Research, to appear.

[8] C. Berge, *Graphs and hypergraphs*, North-Holland, Amsterdam, 1973.

[9] N. Christofides, *Worst-case analysis of a new heuristic for the traveling salesman problem*, Technical Report 388, Graduate School of Indistrual Administration, Carnegie-Mellon University, Pittsburg, 1976.

[10] V. Chvátal, *A greedy heuristic for the set covering problem*, Mathematics of Operations Research **4**, 233-235, 1979.

[11] B. Chandra, H. Karloff and C. Tovey, *New results on the old k-opt algorithm for the traveling salesman problem*, SIAM Journal of computing **28**, 1998-2029, 1999.

[12] S. A. Cook, *The compexity of theorem-proving procedures*, Proc. of the 3rd Annual ACM Symposium on Theory of Computing, 151-158, 1971.

[13] N. Creignou, *Temps linéaire et problèmes NP-complets*, Thèse de doctorat, Université de Caen, France, 1993.

[14] M. Demange, P. Grisoni and V. Th. Paschos, *Differential approximation algorithms for some combinatorial optimization problems*, Theoretical Computer Science **209**, 107-122, 1998.

[15] M. Demange, J. Monnot and V. Th. Paschos, *Bridging gap between standard and differential polynomial approximation : the case of bin-packing*, Applied Mathematics Letters **12**, 127-133, 1999.

[16] M. Demange and V. Th. Paschos, *On an approximation measure founded on the links between optimization and polynomial approximation theory*, Theoretical Computer Science **158**, 117-141, 1996.

[17] M.L. Fisher, G.L. Nemhauser, L.A. Wolsey, *An analysis of approximations for finding a maximum weight hamiltonian circuit*, Operation Research **27** (4), 799-809, 1979.

[18] *M.R. Garey et D.S. Johnson.* Computers and intractability. A guide to the theory of NP-completness. CA.Freeman, San Francisco, 1979.

[19] M. Grigni, E. Koutsoupias and C. Papadimitriou, *An approximation scheme for planar graph TSP*, Proc. of the 36th Annual IEEE Symposium on Foundations of Computer Science, 640-646, 1995.

[20] R. Hassin and S. Khuller, *z-Approximations*, Journal of Algorithms **41**, 429-442, 2001.

[21] J. Håstad, *Clique is hard to approximate within* $n^{1-\varepsilon}$, Proc. of the 37th Annual IEEE Symposium on Foundations of Computer Science, 627-636, 1996.

[22] D.S. Hochbaum éditor, *Approximation algorithms for NP-hard problems*, PWS, Boston, 1997.
chapter 10 : C. Arora and C. Lund, *Hardness of approximation*, 399-446.

[23] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, Plenum Press, New York, 85-103, 1972.

[24] R.M. Karp, *Probabilistic analysis of partitioning algorithms for the TSP in the plane*, Mathematics of Operations Research **2**, 209-224, 1977.

[25] B. Korte and J. Vygen, *Combinatorial Optimization - Theory and Algorithms*, Springer Verlag (2d édition), 2002.

[26] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, *The traveling salesman problem*, Wiley Interscience, New York, 1986.

[27] S. Lin and B.W. Kernighan, *An effective heuristic algorithm for the traveling salesman problem*, Operations Research **21**, 498-516, 1973.

[28] J. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions : a simple polynomial-time approximation scheme for geometric TSP, k-MST and related problems*, SIAM Journal on computing **29**, 1298-1309, 1999.

[29] J. Monnot, *Differential approximations results for the traveling salesman and related problems*, Information Processing Letters **82**, 229-235, 2002.

[30] J. Monnot, V. Paschos and S. Toulouse, *Approximation algorithms for the traveling salesman problem*, Mathematical methods of operations research, **56**, 387-405, 2002.

[31] C. H. Papadimitriou et K. Steiglitz, *Combinatorial Optimization : Algorithms and Complexity*, Prentice Hall, New Jersey, 1981.

[32] C. H. Papadimitriou et S. Vempala, *On the approximability of the traveling salesman problem*, Proc. of the 32nd Annual ACM Symposium the Theory of Computing, 126-133, 2000.

[33] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation and complexity classes*, Proc. of the 20th Annual ACM Symposium the Theory of Computing, 229-234, 1988.

[34] C. H. Papadimitriou and M. Yannakakis, *The traveling salesman problem with distances one and two*, Mathematics of Operations Research **18**(1), 1-11, 1993.

[35] A. Paz and S. Moran, *Non deterministic polynomial optimization problems and their approximation*, Theoretical Computer Science **15**, 251-277, 1981.

[36] A. Schrijver, *On the history of combinatorial optimization (till 1960)*, preprint (available at `http ://homepages.cwi.nl/ lex/`.

[37] *The traveling salesman home page*, `http ://www.math.princeton.edu/tsp/index.html`.

[38] V.V. Vazirani, *Approximations algorithms*, Springer-Verlag, 2001.

[39] E. Zemel, *Measuring yhe quality of approximate solutions to zero-one programming problems*, Mathematics of Operations Research **6**, 319-332, 1981.

Programmation parallèle et systèmes complexes

Responsable : Jean-Louis Roch.

Conférenciers : Jean-Louis Roch, Denis Naddef, Denis Trystram.

Sommaire

Parallélisme, calcul formel et résolution de problèmes complexes.

Denis Naddef[1], Jean-Louis Roch[1], Denis Trystram[1]
[Denis.Naddef@imag.fr, Jean-Louis.Roch@imag.fr,
Denis.Trystram@imag.fr]

Complexité algorithmique et Calcul Formel sont deux domaines très liés. D'une part, la definition de la complexité algorithmique peut être formulée sous un formalisme proche du calcul formel (par exemple pour les circuits arithmétiques). D'autre part, les techniques et outils du calcul formel peuvent être utilisés pour la construction d'algorithmes plus performants.

Ce cours est articulé en 3 parties. La première partie rappelle les principales classes de complexité, séquentielles mais aussi parallèles. La seconde partie est centrée sur la complexité parallèle (classification NC) et sa formulation sous forme de circuits arithmétiques. La technique dite de cascade algorithmique est présentée pour la construction d'algorithmes optimisant plusieurs critères simultanément ; elle est illustrée sur quelques exemples, en particulier la résolution parallèle de systèmes linéaires denses.

Pour faire face à la complexité intrinsèque de certains problèmes, la construction d'algorithmes d'approximation est utilisée ; nous étudions en particulier la résolution du problème de l'ordonnancement crucial en parallélisme.

La dernière partie montre, à travers la résolution du problème du voyageur de commerce (problème de référence en optimisation combinatoire), comment une formulation mathématique duale (en l'occurrence, sous forme de programme linéaire) peut être utilisée pour la résolution d'un problème qui n'admet pas d'approximation polynomiale.

Ce cours est basé principalement sur les documents suivants, dont il reprend certains paragraphes :

- *Complexité parallèle et Algorithmique PRAM*, Jean-Louis Roch, chapitre 5 de l'ouvrage Algorithmes parallèles - Analyse et Conception, G.Authié&al eds, Hermès, 1994
- *Parallel Computer Algebra*, Jean-Louis Roch et Gilles Villard, Tutorial ISSAC 97, Hawai, http ://www-id.imag.fr/ jlroch/perso.html/ps/97-issac.ps.gz
- *Ordonnancement de programmes parallèles sur grappes : théorie versus pratique.*, Jean-Louis Roch, Actes du Congrès International *Algèbre linéaire et Arithmétique : Calcul Numérique, Symbolique et parallèle*, ALA 2001, Université Mohammed V, S. El Hajji éditeur, p. 131–144, 28–31 mai 2001.

[1]Laboratoire ID-IMAG (UMR CNRS-INRIA-INPG-UJF 5132) – ENSIMAG – 51 Av. Jean Kuntzmann, F-38330 Montbonnot Saint-Martin

– *Fondements théoriques pour la conception d'algorithmes efficaces de gestion de ressources*, Olivier Beaumont, Vincent Boudet, Pierre-François Dutot, Yves Robert et Denis Trystram, chapitre d'ouvrage à paraître 2004.
– *Polyhedral Theory and Branch-and-Cut Algorithms for the Symmetric TSP*, Denis Naddef, chapitre 2 de l'ouvrage *The Traveling Salesman Problem and its Variation*, G. Gutin&al. eds, Kluwer Academic Publishers, 2002

1 Complexité parallèle et calcul formel

1.1 Temps parallèle et nombre d'opérations

Soit A un algorithme résolvant un problème P ; on suppose que l'instance P_n de P a $n^{O(1)}$ entrées. Sur le modèle séquentiel RAM, deux caractéristiques (l'une temporelle, l'autre matérielle) sont considérées pour évaluer A :
– le *temps*, noté $T_s(n)$ défini comme le nombre d'opérations effectuées sur des données bornées (ex : opérations flottantes, opérations sur des entiers machines, lecture ou écriture en mémoire d'un mot machine...). La classe $\mathcal{P}$ (resp. $\mathcal{NP}$) est définie comme l'ensemble des problèmes qui admettent un algorithme de temps polynomial $n^{O(1)}$ sur une machine RAM déterministe (resp. non déterministe).
– l'*espace mémoire*, noté $S(n)$ (S pour *space*) défini comme le nombre de places en mémoire nécessaires à l'exécution de l'algorithme.

Par analogie, dans le cadre du calcul parallèle, la qualité d'un algorithme parallèle A résolvant le problème P est basée sur deux caractéristiques, l'une temporelle, l'autre matérielle :
– le *temps parallèle* ou *profondeur* , noté $T_\infty(n)$, qui est le nombre de pas nécessaires à l'exécution de l'algorithme avec un nombre infini de processeurs ;
– le nombre d'opérations $T_1(A)$ effectuées par l'algorithme qui est aussi une borne sur le nombre de processeurs permettant une exécution en temps effectif $T_\infty(n)$.

Il est possible d'exécuter l'algorithme parallèle A sur un nombre de processeurs inférieur à $T_1(A)$, en ordonnançant plusieurs instructions sur un même processeurs. Plus précisément, le "principe de Brent" stipule que, si l'on ne prend pas en compte le surcoût pour la réalisation de l'ordonnancement, l'algorithme parallèle A peut être exécuté sur p processeurs identiques en temps T_p :

$$T_p \leq \frac{T_1}{p} + \left(1 - \frac{1}{p}\right) T_\infty.$$

Ce résultat, dû à Graham, donne aussi une méthode pour la construction d'un tel ordonnancement.

1.2 La classification NC

Une question de base en calcul parallèle est de déterminer les problèmes intrinsèquement parallèles, c'est-à-dire qui peuvent être résolus beaucoup plus rapidement avec plusieurs processeurs plutôt qu'avec un seul [6] [15].
La classe *NC*, formalisée par Nicholas Pippenger [21] (et nommée par Cook *NC* pour "Nick's class" [6]), est la classe des problèmes qui peuvent être résolus en temps poly-logarithmique (c'est-à-dire résolues plus rapidement qu'il ne faut de temps pour lire séquentiellement leurs entrées) sur une machine parallèle ayant un nombre polynomial de processeurs, autrement dit de surface raisonnable. *NC* peut donc être caractérisée par :

$$\mathcal{NC} = \{ \text{ problèmes } P / \exists A \text{ algorithme qui résout } P_n \text{ en coût } T_\infty(n) = \log^{O(1)} n \text{ et } T_1(n) = n^{O(1)} \}\}.$$

NC est trivialement un sous-ensemble de la classe *P* des fonctions qui peuvent être calculées séquentiellement en temps polynomial. Ainsi $P \supseteq NC$ mais l'inclusion stricte reste conjecturelle.
Une propriété fondamentale de *NC* est d'être **résistante** : elle reste la même quel que soit le modèle parallèle, par exemple PRAM, circuits, etc.

Remarque. Classes probabilistes. Comme en séquentiel, le préfixe R (resp. Z) désigne une classe de complexité probabiliste de type Monte Carlo – R pour Random – (resp. de type Las Vegas – Z pour Zero-error –). Ainsi la classe $\mathcal{RNC}$ est l'ensemble des problèmes pouvant être résolus par un algorithme Monte Carlo de temps parallèle poly-logarithmique et effectuant un nombre polynômial d'opérations.

1.3 Modèle booléen et NC-Réduction

De façon à pouvoir classer les problèmes par ordre de difficulté à l'intérieur de NC, et préciser où peut se trouver la différence entre P et NC, une relation d'ordre entre les problèmes est définie dans le cadre du modèle booléen [2] : la NC-réductibilité.

1.3.1 Le modèle booléen

Dans ce modèle, une machine parallèle est une famille *uniforme* $(B_n)_{n \in I\!N}$ de circuits booléens (i.e. graphes orientés et sans cycle – DAG –) telle que B_n a $n^{O(1)}$ entrées. Un nœud du circuit est ici une porte logique effectuant une opération booléenne (et, ou, négation). On distingue essentiellement deux sous-modèles,

selon que le nombre d'entrées d'une porte (fan-in) est borné (i.e. vaut 2 ici) ou non borné. Le nombre de sorties d'une porte (fan-out) est quant à lui non borné [1]. Les nÏuds d'entrée (respectivement de sortie) ont un fan-in (respectivement fan-out) de 0. L'uniformité permet de limiter la complexité architecturale du circuit [24]. On utilise le plus souvent la "log-uniformité" qui signifie que la description du circuit B_n (pour $n \in I\!N$) peut être calculée sur une machine de Turing avec un espace logarithmique.
La surface $T_1(n)$ est ici définie comme le nombre de nœuds du circuit B_n, et le temps parallèle $T_\infty(n)$ comme sa profondeur.
Dans ce modèle on distingue les sous-classes NC^k (respectivement AC^k) pour les circuits dont les portes ont un fan-in borné (resp. non borné) et qui sont de profondeur $O(\log^k n)$ et de taille $n^{O(1)}$. On a alors les relations suivantes [15] :

$$NC^k =\subseteq AC^k \subseteq NC^{k+1}$$

La classe NC est alors définie comme l'union de toutes les classes NC^k :

$$NC = \bigcup_{k=0}^{\infty} NC^k$$

Remarque : circuits arithmétiques. Des extensions du modèle booléen [8] permettent de considérer que les portes du circuit peuvent faire en temps unité des opérations sur un espace donné E (par exemple les rationnels, ou les polynômes à coefficients rationnels). Les classes de complexité correspondantes sont alors notées NC^k_E pour préciser que les opérations de base considérées sont des opérations sur E. Par exemple, le produit de n entiers de n bits appartient à $NC^1_{I\!N}$ (produit itéré) mais le même algorithme ne permet que de prouver l'appartenance à NC^2 (la multiplication de deux entiers de n bits appartenant à NC^1).

1.3.2 NC-réductibilité et problèmes P-complets.

Une fois le modèle booléen défini, il est maintenant possible de classifier les problèmes selon leur complexité, grâce à une relation d'ordre : la NC^1-réductibilité [6]. On dit qu'une fonction (ou un problème) f est NC^1-réductible à une fonction g (ce qui est note $f \leq_{NC^1} g$) s'il existe une famille uniforme de circuits qui calcule f en temps logarithmique ($O(\log n)$), et dont les nœuds sont soit des portes booléennes, soit des oracles permettant de calculer g. Un oracle pour g est ici un nœud ayant r entrées $(e_1, \ldots, e_r)$ et t sorties $(s_1, \ldots, s_t)$ et qui calcule le résultat $(s_1, \ldots, s_t) = g(e_1, \ldots, e_r)$. La profondeur d'un tel nœud est assimilée à

[1] Un circuit de fan-in borné et de fan-out non borné peut en effet être transformé en un circuit de même surface et de même temps -en ordre- qui soit de fan-in et de fan-out borné [13].

$\log(rt)$.
Il est clair que la relation de NC^1-réductibilité est réflexive et transitive et que NC^k est close par NC^1-réductibilité.
Soit E une classe de problèmes. On dira qu'une fonction (un problème) f est $NC^1 - dur$ pour l'ensemble E (ou E-dur) si et seulement si :

$$\forall g \in E : g \leq_{NC^1} f$$

f est dit complet pour E (E-complet) si f est E-dur et si $f \in E$.
Nous avons vu que $NC \subseteq P$. L'inclusion stricte restant conjecturelle, on peut se demander quels sont les problèmes complets pour P, qui sont ceux contenant -ou susceptibles de contenir- le moins de parallélisme intrinsèque.
Le problème P-complet de référence (l'analogue de la satisfaisabilité pour la complexité séquentielle et la classe NP) est le MCVP (monotone circuit value problem) [10] : " Etant donné une séquence de n équations booléennes du type $e_1 = 0$, $e_2 = 1$ et $e_k = e_i \wedge e_j$ ou $e_k = e_i \vee e_j$ pour $1 \leq i \leq j < k \leq n$, calculer la valeur de e_n [2]."

1.4 Evaluation de circuits arithmétiques

Nous avons vu que tout problème dans P pouvait être réduit au problème MCVP, qui est P-complet. Toutes les techniques permettant d'évaluer rapidement en parallèle des instances du MCVP pourront donc être appliquées à n'importe quel problème polynômial (pour autant que l'on puisse construire "facilement" les instances du MCVP correspondant au problème que l'on cherche à paralléliser)[22].
De manière générale, une instance du MCVP se présente comme un programme arithmétique sans boucle, de longueur n, (i.e. un graphe de précédence comportant n nœuds) ne comportant que des affectations ou des opérations booléennes $\wedge$ ou $\vee$.
Différentes techniques ont été proposées pour évaluer rapidement des programmes sans boucles dans des structures algébriques.
En particulier, l'évaluation d'expression arithmétique dans un anneau ou un corps appartient à NC [4] [9]. Ainsi, si le DAG correspondant au programme sans boucle peut être transformé en un arbre ayant $m = n^{O(1)}$ nœuds, alors une évaluation parallèle de cet arbre permet de l'évaluer en temps $T_\infty = O(\log m)$ avec $T_1 = \Theta(m)$ opérations.
Mais si le nombre de nœuds de l'arbre correspondant à l'expansion du circuit est exponentiel, l'évaluation ne pourra se faire en parallèle qu'en temps linéaire . Comme exemple, on peut considérer le programme suivant : $x_i := x_{i-1} + x_{i-1}$

[2] Tout problème s'exécutant en temps polynômial sur une machine de Turing déterministe peut être NC^1-réduit à ce problème [15], ce qui prouve, étant clairement dans P, qu'il est P-complet.

pour $i = 1, \ldots, n$ avec comme entrée x_0 un entier dans le monoïde $(I\!N, +)$. Pour pallier à ce problème, il est cependant possible d'utiliser la puissance de la structure algébrique de l'ensemble dans lequel est défini le problème. Pour l'exemple précédent, il est clair que le programme calcul $2^n.x$; l'existence de la multiplication, distributive par rapport à l'addition, peut permettre de ramener le problème à un produit itéré pour calculer 2^n. Plus précisément, il est possible d'évaluer plus rapidement le circuit en tirant parti à la fois de l'associativité et de la distributivité d'une loi par rapport à l'autre ; ce résultat a été donné par Kaltofen, Miller et Ramachandran dans [14] pour un anneau (i.e. les opérations $+$ et $\times$ du programme sont dans un anneau) et étendu dans [22] au cas des treillis, en particulier des booléens cadre du problème P-complet de référence (MCVP).
On définit le degré arithmétique d d'un circuit (DAG correspondant à un programme sans boucle) comme suit. le degré d'une entrée est 1 ; le degré d'un nœud $+$ le maximum des degrés de ses opérandes ; le degré d'un nœud $\times$ la somme des degrés de ses opérandes. Le degré du circuit est le maximum du degré de ses nœuds.
Dans la proposition ci-dessous, $M(n)$ est le nombre d'opérations nécessaires pour effectuer un produit de matrices en temps $\log n$ – $M(n) < n^3$ –.

Proposition 1 *[14] Tout programme sans-boucle de n nÏuds et de degré d dans un semi-anneau peut être évalué en temps $T_\infty(n) = O(\log n \log(n.d))$ avec $T_1(n) = O(M(n) \log n \log(n.d))$ opérations.*

Exemple : résolution de système triangulaire. Considérons en exemple la résolution du système linéaire triangulaire inversible $Ax = b$ par la technique classique d'élimination itérative dite de descente triangulaire. Soit n la dimension de A. Le circuit associé à ce programme sans boucle comporte $\Theta(n^2)$ nÏuds. Son degré est celui du nÏud associé à x_n ; or le degré de x_k est le degré de $x_{k-1}+O(1)$, pour tout k. Le degré du programme est donc $\Theta(n)$. On en déduit que la résolution d'un système linéaire peut être réalisée en temps $T_\infty(n) = O(\log^2 n)$ avec $T_1(n) = M(n^2) = O(n^6)$ processeurs. Ce problème appartient donc à NC^2 ; mais le nombre important d'opérations interdit son utilisation pratique.

Remarque : cas des treillis. Des extensions de ce résultat à la structure algébrique de treillis ont été données [22]. Cette structure est intéressante, car elle permet de mieux prendre en compte la structure des booléens, cadre du MCVP.

1.5 Algorithmes en cascade

Par rapport à un algorithme séquentiel, introduire du parallélisme nécessite l'introduction d'opérations supplémentaires afin d'obtenir un algorithme de temps

parallèle T_∞ minimal, quitte à augmenter le nombre d'opérations T_1 requis par rapport à T_s..
Cependant, lors de l'exécution, on ne peut utiliser qu'un nombre limité de ressources. Le problème est alors de replier efficacement le parallélisme. Le principe de Brent motive la construction d'algorithmes parallèles qui minimisent donc T_∞ tout en gardant T_1 proche du nombre d'opérations du meilleur algorithme séquentiel possible.
Nous présentons deux solutions pour effectuer un tel repliage de l'algorithme parallèle. La première est basée sur la construction d'ordonnancements efficaces ; c'est l'objet de la section suivante. La deuxième est algorithmique ; elle est basée sur un couplage de plusieurs algorithmes, généralement au moins un algorithme séquentiel et un algorithme parallèle. Cette technique est appelée *cascade algorithmique*. Nous l'illustrons sur trois exemples : maximum de n éléments, inversion de matrices et découpe récursive à grain adaptatif.

1.5.1 Circuit à fan-in non borné pour le calcul du maximum

Avec un circuit à fan-in non borné, le maximum de n éléments peut être calculé par -entre autres- les 3 algorithmes suivants :
- calcul séquentiel itératif classique : $T_1 = n$; $T_\infty = n$
- calcul parallèle récursif en arbre d'arité K (par exemple $K = 2$ ou $K = \sqrt{n}$) : $T_1 = n$; $T_\infty = \log n$
- calcul parallèle avec comparaison parallèle de chaque élément à tous les autres ; seul l'élément trouvé supérieur à tous les autres est gardé en temps constant avec une porte "ET-logique" n-aire. $T_1 = n^2$; $T_\infty = O(1)$

L'algorithme qui minimise T_∞ (le troisième) demande un accroissement important du nombre d'opérations T_1.
Le couplage de ces 3 algorithmes différents permet la construction d'un algorithme qu vérifie $T_\infty = \log\log n$ tout en gardant $T_1 = \Theta(n)$.

Transparents ; transparents 12–20, pages 3,4 du document :
`http ://www-id.imag.fr/~jlroch/perso.html/COURS/2003-dea-algo-par/2001-10-22-cours2.pdf`

1.5.2 Résolution de systèmes triangulaires

La technique d'évaluation du circuit (§1.4) a montré qu'il était possible de résoudre en temps $T_\infty = \log^2 n$ un système triangulaire ; le problème est ici de trouver un algorithme parallèle qui effectue le même nombre d'opérations qu'un algorithme séquentiel, i.e. $T_1 = n^2$.

Analyse du parallélisme du meilleur algorithme séquentiel L'algorithme de résolution par descente triangulaire a un coût $T_s(n) = \Theta(n^2)$. L'analyse des dépendances de cet algorithme (fig. 1) conduit au coût parallèle : $T_\infty = n$ et $T_1 = n^2$.

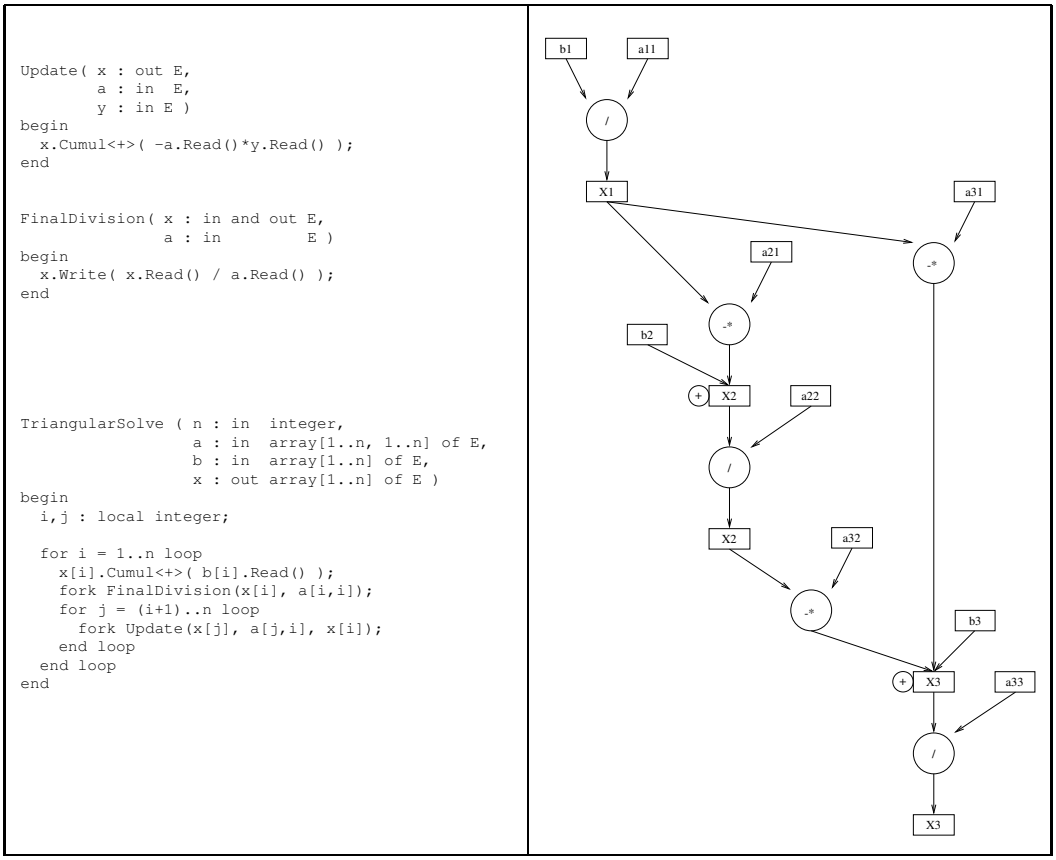


FIG. 1 – Graphe de dépendance pour la résolution d'un système triangulaire inversible de dimension 3.

Nous avons vu que l'évaluation par circuit arithmétique conduisait à un temps parallèle $T_\infty = \log^2 n$. On considère ici l'algorithme récursif [3]. Soient A, b et x partitionnés comme suit :

$$A = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{1}$$

Ici A_{11} est de taille $h \times h$, x_1 et x_2 sont de taille h. On a :

$$A_{11}x_1 = b_1 \quad \text{et} \quad A_{22}x_2 = b_2 - A_{21}x_1. \tag{2}$$

où x_1 et x_2 sont calculés récursivement en utilisant le même algorithme ; $A_{21}x_1$ est calculé par un produit scalaire.

Supposons que l'on stoppe la découpe récursive quand les matrices sont de taille $k \times k$, et que l'on utilise alors un algorithme séquentiel pour les inversions de système triangulaire et produits matrice-vecteur dès que le système ou la matrice est de dimension inférieure à une valeur k. Le coût de l'algorithme parallèle résultant est alors :

$$T_\infty = \Theta(n.k) \quad T_1 = \Theta(n^2) \tag{3}$$

Augmenter le parallélisme La dépendance entre x_1 et x_2 dans 2 peut être éliminée en calculant directement en parallèle les inverses des matrices inversibles A_{11} et A_{22}.
Considérons la matrice A partitionnée en quatre blocs de dimension $n/2$ (1 avec $h = n/2$). On a alors :

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{22}^{-1} A_{21} A_{11}^{-1} & A_{22}^{-1} \end{bmatrix} \tag{4}$$

Dans la suite, nous considérons que le produit de deux matrices de dimension n peut être calculé en $T_\infty = \log n$ avec $T_1 = O(n^3)$ opérations.
Selon 4, l'inverse de A peut être calculée comme suit ; on calcule d'abord en parallèle A_{11}^{-1} et A_{22}^{-1} ; puis on calcule le dernier bloc A_{21}^{-1} de A^{-1} en effectuant séquentiellement deux produits paralèles de matrices. Le coût d'inversion de A est alors $T_\infty = \log^2 n$ avec $T_1 = O(n^3)$ opérations. Une fois A^{-1} calculée, on peut calculer $x = A^{-1}b$ avec un coût négligeable devant celui de l'inversion de A.
Cependant, même si de coût polylogarithmique, cet algorithme effectue n fois plus d'opérations que l'algorithme séquentiel. Dans le paragraphe suivant, l'utilisation de la technique de cascade permet de limiter ce surcoût en nombre d'opérations.

Remarque. On peut remarquer qu'en utilisant un produit rapide de matrices, le nombre d'opérations devient $T_1 = n^\omega$ avec $\omega < 2.38$ [20].

Résolution de système linéaire en cascade L'algorithme précédent n'est pas efficace mais peut être couplé à l'algorithme séquentiel récursif par vloc pour l'accélérer (formule 2). En effet, il peut être utilisé sur des matrices de petite dimension (disons inférieure à h) lorsque le surcoût $O(h^3)$ dû à l'inversion rapide de ce bloc devient négligeable par rapport au coût de la mise à jour suite à l'inversion du bloc (environ nh).

Théorème 2 *La solution d'un système traiangulaire inversible peut être calculée en $T_\infty = n^{1/2}$ et $T_1 = n^2$) en utilisant un algorithme de multiplication de matrices effectuant. n^3 opérations.*
Si un algorithme en n^ω est utilisé pour la multiplication, le coût devient : $T_\infty = n^{(\omega-2)/(\omega-1)} \log^2 n$ et $T_1 = n^2$

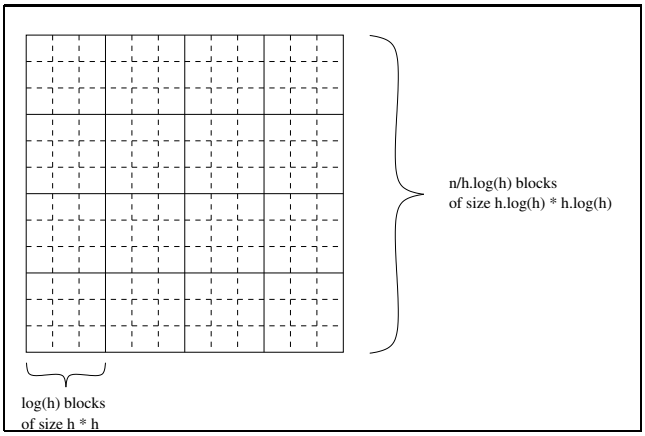


FIG. 2 – Partitionnement avec $h = 8$, $h \log h = 24$, $n = 96$

Selon 3, partitionnons A en n^2/h^2 blocs de taille $h \times h$. Un calcul direct (cf théorème 3) conduit à un temps paralèle $T_\infty = O(n^{1/2} \log^2 n)$. Pour éviter le facteur de surcoût $\log n$ overhead factor in the parallel time, on regroupe les calculs sur des blocs de dimension $\log^2 h$.
Soit $k = h \log h$; la matrice A peut être découpée en $(n/k)^2$ blocs, chaque bloc contenant $\log^2 h$ sous-blocs de dimension h (cf fig. 2).
On utilise l'algorithme itératif sur les matrices de taille $(n/k) \times (n/k)$
A l'étape i, nous devons inverser le système triangulaire correspondant au bloc diagonal (i, i). Pour ce calcul, on inverse d'abord les $\log h$ diagonales de ce bloc. Ensuite, on met à jour les autres sous blocs liés à x_i. A la fin de cette étape i, les blocs x_j, pour $j > i$, sont mises à jour. L'algorithme est le suivat :

Initialisation.

On partitionne A en n/k blocs $M_{i,j}$ de dimension k ($k = h \log h$). Pour $1 \leq j \leq i \leq n/k$, on partitionne $M_{i,j}$ en $\log h \times \log h$ blocs $m_{i,j}^{k,l}$ de dimension h.

Soit x intialisé à b et partitionné selon A.

for $i = 1..n/k$ do

1. for $j = 1.. \log h$ do
   fork $\left(m_{i,i}^{j,j}\right)^{-1} = \text{invert}(m_{i,i}^{j,j})$.
   En utilisant l'inevrsion et le principe de Brent, le coût est $T_\infty = \log^2 h$ et $T_1 =, h^3 \log h$.
2. for $j = 1.. \log h$ do
   update x_i^j in parallel
   $x_i^j = \left(m_{i,i}^{j,j}\right)^{-1} \left(x_i^j - \sum_{l=1}^{j-1} m_{i,i}^{j,l} x_i^l\right)$

Les produits scalaires sont effecués en parallèle : ainsi, x_i est calculé avec un coût $T_\infty = \log^2 h$ et $T_1 = h^2 \log h$).

3. for $j = i + 1..n/k$ fork update x_j in parallel
   $x_j = x_j - M_{j,i} x_i$
   En effectuant les produits scalaires en parallèle, le coût est $T_\infty = \log h$ et $T_1 = nh \log h$).

Finalement, le coût de cet algorithme en cascade est $T_\infty = n \log^2 h/k$ et le nombre d'opérations est $T_1 = n/k \max(h^3 \log h, nh \log h))$. Avec $k = h \log h$ et choisissant $h = n^{1/2}$, on ontient

$$T_\infty = n^{O,5} \quad T_1 = n^2$$

La même technique est appliquée pour obtenir la borne supérieure lorsqu'un algorithme de multiplication rapide est utilisé. □
Dans les exemples précédents, la cascade était basée initialement sur un algorithme tres parallele qui effectuait trop d'opérations ; l'objectif de la cascade était alors de réduire le nombre d'opérations en cascadant cet algorithme parallèle sur un algorithme effectuant moins d'opérations – typiquement un algorithme séquentiel –.
La section suivante montre une manière duale de réaliser la cascade : en cascadant un algorithme séquentiel sur un algorithme parallèle pour réduire cette fois le temps parallèle.

1.5.3 Granularité adaptive

Cette technique est décrite dans [16] : elle consiste à cascader à utiliser un algorithme parallèle pour accélérer un algorithme séquentiel.
Nous illustrons brièvement cette technique, duale de la précédente, sur la résolution du système linéaire précédent. Par défaut, en supposant le nombre de ressources faible, le programme est démarré selon un algorithme séquentiel par bloc. Si un processeur est inactif, il cherche à extraire des calculs pouvant être effectués en parallèle chez un processeur actif. Cette extraction peut être réalisée en accédant aux prochains blocs qui doivent être traités par l'algorithme séquentiel.
Ainsi, l'algorithme séquentiel dégénère en exécution parallèle. Cette dégénération est récursive, et ne limite donc pas le parallélisme : l'extraction peut être effectuée jusqu'à des blocs de dimension minimale.
Ce couplage, qui réalise une cascade dynamiquement en fonction de l'inactivité des processeurs, est plus général que la cascade précédente qui était statique. On obtient ainsi un algorithme dont le temps d'exécution sur un processeur est garanti le même que le temps de l'exécution du meilleur algorithme séquentiels ; et sur un grand nombre de processeurs, on obtient un temps d'exécution parallèle minimal.

2 Ordonnancement et algorithmes d'approximation

Après une description du problème d'ordonnancement et de sa complexité, nous montrons qu'il existe des algorithmes d'approximation pour résoudre certaines instances.
Nous montrons ensuite comment un ordonnancement peut être construit pour optimiser plusieurs critères, comme temps parallèle, nombre d'opérations et espace mémoire avec le vol de travail.
Dans cette section, nous allons illustrer la démarche générale sur un des problèmes les plus simples en ordonnancement, la distribution de tâches sur des machines homogènes, sans prendre en compte le coût des communications.

2.1 Problème d'ordonnancement et complexité

La plupart des problèmes d'allocation de ressources s'expriment comme des problèmes d'optimisation combinatoire opérant sur des graphes (minimisation ou maximisation de certains critères sous contraintes). Ainsi, il est classique de chercher à déterminer une exécution de durée minimale d'un programme à paralléliser représenté par un graphe orienté acyclique (DAG, directed acyclic graph) $G = (V, E)$ [5] appelé graphe de tâches ou graphe de précédence.
Dans le graphe G, les sommets représentent les différentes tâches à traiter, généralement pondérées par leur temps d'exécution. Une arête (orientée) entre les tâches i et j signifie que la tâche i doit être traitée avant le début de la tâche j (ce qui justifie le caractère acyclique du graphe). Quand on considère des coûts de communication, l'arête (i, j) est généralement pondérée par le temps de communication nécessaire si les tâches i et j ne sont affectées au même processeur. Sous forme de problème de décision, le problème que nous allons considérer (problème central de l'ordonnancement multiprocesseur - PCOM) peut s'écrire de la manière suivante :
PCOM :
Instance : Soit $G = (V, E)$ un graphe d'ordre n (dont E est codé sous forme de matrice d'adjacence par exemple), on note p_j la pondération du sommet j (c'est-à-dire la durée d'exécution de la tâche j) et un ensemble de m processeurs identiques complètement connectés (logiquement) par un réseau homogène (et infiniment rapide). On se donne également une borne k (un entier) sur la durée d'exécution.
Question. Existe-t-il un ordonnancement valide des n tâches en moins de k unités de temps ?
On cherche donc une application σ qui à chaque tâche de V associe la date où elle débutera son exécution en respectant les contraintes de précédence de E et une application π qui à chaque tâche de V associe le processeur où elle s'exécutera.

Mathématiquement, ce problème s'exprime de la façon suivante : $\forall j \in V$, déterminer σ de V dans $\mathbb{N}$ et π de V dans $\{1; m\}$ telle que si $(i, j) \in E$ alors $\sigma(j) \geq \sigma(i) + p_i$ et pour tout couple (i, j) si $\pi(i) = \pi(j)$ alors on a soit $\sigma(j) \geq \sigma(i) + p_i$, soit $\sigma(i) \geq \sigma(j) + p_j$.

Théorème 3 *Le problème PCOM est NP-complet.*

La démonstration de ce résultat s'obtient assez facilement par une réduction à partir du problème 3-Partition. Il est NP-complet au sens fort pour m arbitraire et le reste pour m fixé.
Bien que ce problème soit NP-dur, il existe des algorithmes d'approximation efficaces. Nous étudions dans la section suivante l'ordonnancement glouton qui donne une preuve constructive du principe de Brent (§1.1).

2.2 Exemple d'algorithmes d'approximation

Sur machine parallèle, les ordonnancements classiquement utilisés sont de type liste : ils consistent à affecter à un processeur inactif une tâche prête à être exécutée si il en existe. En effet, en supposant $T_\infty \ll T_1$ et si on néglige le coût de l'interprétation, un tel algorithme fournit un temps d'exécution asymptotiquement optimal comme le montre le résultat suivant. Ce théorème est important ; preuve constructive du principe de Brent (§1.1), il justifie le fait de s'intéresser à des algorithmes parallèles de temps critique très faible.

Théorème 4 *[12] Si l'on ne prend pas en compte le coût de calcul et de réalisation de l'ordonnancement, tout ordonnancement de type liste appliqué à un programme parallèle de coût T_1 et T_∞ conduit à un temps d'exécution T_p sur p processeurs majoré par :*

$$T_p \leq \frac{T_1}{p} + T_\infty$$

Preuve Nous rappelons la preuve car son schéma est utilisé pour l'analyse de nombreuses variantes.
Soit t_1 une tâche qui s'est terminée à la date T_p, et soit d_1 la date du début de cette tâche. Avant d_1, deux situations peuvent être distinguées : i) soit aucun processeur n'a été inactif avant d_1 ; ii) soit au contraire il existe une date $d < d_1$ à laquelle au moins un processeur était inactif.
Dans ce cas ii), soit d' la plus grande de ces dates. L'ordonnancement étant de liste, si à la date d' la tâche t_1 avait été prête alors elle aurait débuté son exécution. Il existe donc une tâche t_2 en cours d'exécution à la date d' telle que $t_2 \prec_G t_1$. Soit d_2 la date de début d'exécution de cette tâche.

L'application récursive de ce schéma permet de construire une séquence $t_k \prec_G \cdots \prec_G t_2 \prec_G t_1$ de tâches telles qu'à tout instant de l'exécution, soit tous les processeurs sont actifs, soit un des processeurs est en train d'exécuter une des tâches de cette séquence. La durée de la première situation est majorée par $\frac{T_1}{p}$ et celle de la seconde par la durée d'un chemin critique du graphe, c'est-à-dire par T_∞. □

Par une preuve similaire [17], on montre que, sur m machines, un ordonnancement de liste est une approximation à un facteur $\left(2 - \frac{1}{m}\right)$ de l'ordonnancement optimal sur m machines. Dans la section suivante, nous montrons que ce ratio de performance est optimal.

2.3 Bornes inférieures sur le ratio de performance

Une question naturelle est alors de déterminer s'il est possible d'obtenir un ratio de performance inférieur à $\left(2 - \frac{1}{m}\right)$, soit sur le même problème soit en utilisant des opérations plus puissantes pour réaliser l'ordonnancement, comme le tirage aléatoire ou la migration. Ce problème est étudié dans [25] où la proposition suivante est prouvée.

Théorème 5 *[25] Si les temps de communication ne sont pas pris en compte,* $\left(2 - \frac{1}{m}\right)$ *est une borne inférieure pour le ratio d'un algorithme d'ordonnancement déterministe avec ou sans migration.*
Un algorithme probabiliste sans préemption ne peut avoir un ratio de performance inférieur à $\left(2 - \frac{1}{\sqrt{m}}\right)$*, ratio qui est obtenu par un algorithme de liste dans lequel les tâches affectées aux processeurs inactifs sont tirées au hasard parmi l'ensemble des tâches prêtes.*

Seule la preuve de la première partie est ici présentée car elle utilise la construction d'un adversaire, technique fréquemment utilisée pour obtenir des bornes inférieures sur un algorithme à la volée. Il s'agit de construire une instance du problème réalisant le pire cas et de montrer que ce pire cas ne peut être évité par aucun algorithme d'ordonnancement.
Le pire cas est obtenu pour l'instance suivante due à Graham [11]. G contient $1 + p(p-1)$ tâches indépendantes ; une tâche α est de longueur p tandis que les β_k, $1 \leq k \leq p(p-1)$ autres sont de longueur 1. L'ordonnancement optimal est de longueur p ; il est obtenu en exécutant α_1 sur un processeur et les $p(p-1)$ tâches β_k sur les $p-1$ autres processeurs.
Aucun algorithme d'ordonnancement à la volée ne peut faire d'hypothèse sur la longueur d'une tâche qui est inconnue tant que la tâche n'est pas terminée. La technique de l'adversaire consiste alors à faire en sorte que la la tâche α soit démarrée le plus tard possible. Chaque fois que l'ordonnanceur lance l'exécution

d'une tâche sur un processeur, nous supposerons donc que cette tâche est une tâche β_k, de longueur 1. Ainsi l'ordonnanceur ne peut demander l'exécution de la tâche α qu'après que toutes les tâches β_k aient été exécutées, donc au plus tôt au top $(p-1)$. La longueur de l'ordonnancement délivré est alors de longueur au moins $(p-1)+p$, ce qui montre la borne inférieure. Il est clair que la préemption ou la migration ne peuvent améliorer ce ratio. □.
En conséquence, ni la migration ni l'introduction d'aléatoire ne peuvent améliorer le ratio de performance en comparaison d'un algorithme de liste.
Pour un programme à grain fin (T_∞ petit), un algorithme de type liste apparaît donc pertinent puisque asymptotiquement optimal. Cependant, la preuve ne prend pas en compte le coût de gestion de calcul de l'ordonnancement, à savoir la gestion de la liste des tâches prêtes. Ce problème est étudié dans la section suivante avec les algorithmes de vol de travail.

2.4 Ordonnancement par vol de travail (work-stealing)

D'un point de vue performance effective, le coût de la réalisation d'un ordonnancement doit aussi être pris en compte [17]. Pour un algorithme de liste, il est a priori borné par le nombre n de tâches ; comme $n > \frac{T_1}{T_\infty}$, le surcoût peut être considérable pour un programme de grain fin.
Toutefois, la preuve précédente montre que le nombre de tops d'inactivité est majoré par T_∞ sur chaque processeur ; ainsi, si les processeurs sont capables de trouver facilement des tâches prêtes à exécuter lorsqu'il en existe, le surcoût d'ordonnancement, dans ce cas majoré par $O(pT_\infty)$, sera alors négligeable pour des programmes possédant un grand degré de parallélisme.
Ce constat est à la base de la stratégie "travail d'abord" ("work-first principle" [7]), traditionnellement utilisée pour la compilation des langages fonctionnels [18]. Cette stratégie elle consiste à mettre la plus grande partie du surcoût de réalisation de l'ordonnancement sur le chemin critique, i.e. lorsqu'un processeur devenu inactif doit voler une tâche à un autre processeur qui possède des tâches prêtes. Par rapport à la preuve du théorème 4, le surcoût est mis sur le terme en T_∞ plutôt que sur celui en T_1/p. Ainsi, pour des programmes très parallèles où T_∞ est négligeable devant T_1, le surcoût dû à l'ordonnancement est moindre.
La solution est donc de n'effectuer la gestion locale de tâches que lorsque cela est nécessaire, à savoir lors d'un vol. Pour cela, la création d'une tâche est compilée en appel de fonction locale. Cela nécessite une hypothèse fondamentale : toute tâche créée doit pouvoir être directement exécutée localement, donc être prête. Cette hypothèse est facilement vérifiée pour les langages offrant un parallélisme de type série-parallèle, comme les langages fonctionnels qui ont été les premiers à l'exploiter. Elle n'est cependant pas restrictive à ce type de parallélisme et est implantée dans des langages plus généraux comme Athapascan [23].

Le programme est alors *dégénéré en exécution séquentielle profondeur d'abord*. Le point critique est que cette dégénérescence ne doit pas entraîner de perte de parallélisme au niveau de l'algorithme.
La dégénération séquentielle est basée sur un ordre total sur les tâches, compatible avec l'ordre partiel défini par la relation de précédence. Cet ordre total est celui suivi par l'exécution séquentielle, par exemple profondeur d'abord. La section suivante montre que le choix de cet ordre peut être fait de manière à optimiser l'espace mémoire requis.

2.5 Optimisation sous contraintes : temps parallèle et espace mémoire

Sur un nombre fini de ressources, optimiser l'espace mémoire conduit à un ordonnancement séquentiel alors que minimiser le temps d'exécution conduit à exploiter au mieux le parallélisme ; les deux objectifs, mémoire et temps, sont donc antagonistes.
Pourtant, il est possible de construire des ordonnancements qui sont asymptotiquement optimaux en temps (i.e. si T_∞ est asymptotiquement négligeable devant T_1) et qui garantisse simultanément un espace mémoire borné par rapport à une exécution séquentielle.
Plus précisément, l'ordonnancement séquentiel du programme peut lui aussi influencer conséquemment sur la consommation mémoire. Considérons par exemple un programme comportant n tâches a_i allouant 1 mot chacune et n tâches b_i effectuant les libérations correspondantes ; les seules contraintes de précédence soient du type $a_i \prec l_i$. Considérons les deux ordonnancements séquentiels suivants :

$$a_1 < b_1 < \ldots < a_i < b_i < \ldots < a_n < b_n \quad (1)$$
$$a_1 < \ldots < a_i < \ldots < a_n < b_1 < \ldots < b_i < \ldots < b_n \quad (2)$$

Le volume mémoire requis pour l'exécution correspondant à l'ordre (1) est de 1 mot, tandis que celui correspondant à l'ordre (2) est de n mots. Le volume mémoire requis dépendant fortement de l'ordre, le principe pour contrôler cette consommation lors d'une exécution parallèle est de suivre l'ordre séquentiel dit de *référence* qui a été utilisé pour définir l'espace mémoire séquentiel S_1 qui sert de référence [1, 19]. Il est alors possible de comparer S_p à S_1.

Théorème 6 *[1] Un ordonnancement de liste qui, parmi les tâches prêtes, affecte à un processeur inactif la plus prioritaire selon l'ordre séquentiel de référence requiert un espace mémoire s majoré par :*

$$S_p \leq S_1 + pKT_\infty.$$

où S_1 est l'espace mémoire requis par l'exécution séquentielle de référence et K l'espace mémoire maximum alloué par une tâche.

Preuve Les tâches restant à exécuter sont stockées dans une liste triée selon l'ordre d'une exécution séquentielle[3]. Lorsqu'un processeur termine une tâche, il prend la tâche en tête de cette liste pour l'exécuter. Les tâches les plus vieilles selon l'ordre séquentiel sont donc prioritaires. La durée d'une tâche étant majorée par T_∞, à chaque instant au plus T_∞ tâches peuvent être exécutées en avance par rapport à l'ordre séquentiel sur chaque processeur. Chaque tâche allouant au plus un espace mémoire de K, l'espace mémoire total requis est majoré par $S_1 + pKT_\infty$. □
Aussi, la gestion de l'ordre des tâches dans la liste est crucial pour les performances. Dans le contexte de machines à mémoire partagée (SMP), Cilk utilise un tel ordonnancement. Dans un contexte distribué, Athapascan implémente une gestion distribuée d'une pile cactus qui garantit l'espace mémoire utilisé.

2.6 Construction d'ordonnancements multi-critères

Usually, approximation algorithms are designed with respect to one criterion. However, several criteria could be used to describe the quality of a schedule. In the context of parallel processing, the choice of which criterion to choose depends on the priorities of the users.
However, one could wish to take advantage of several criteria in a single schedule. With the makespan and the sum of weighted completion times, it is easy to find examples where there is no schedule reaching the optimal value for both criteria. Therefore you can not have the cake and eat it, but you can still try to find for a schedule how far the solution is from the optimal one for each criterion. In this section, we will look at a generic way design algorithms with guarantees on two criteria and at a more specific algorithm family for the moldable case. More details can be found in the chapter of Dutot, Mounie and Trystram in the Handbook of Scheduling edited by Joseph Leung (April 2004).

Two Phases, Two Algorithms ($\mathcal{A}_{\sum C_i}$, $\mathcal{A}_{C_{max}}$) Let us use two known algorithms $\mathcal{A}_{\sum C_i}$ and $\mathcal{A}_{C_{max}}$ with performance ratios respectively $\rho_{\sum C_i}$ and $\rho_{C_{max}}$ with respect to the sum of completion time and the makespan.

Proposition. It is possible to combine $\mathcal{A}_{\sum C_i}$ and $\mathcal{A}_{C_{max}}$ in a new algorithm with a performance ratio of $2\rho_{\sum C_i}$ and $2\rho_{C_{max}}$ at the same time.

[3]L'insertion et la suppression dans la liste peuvent être effectuées en un nombre constant d'opérations par chaînage : les tâches filles sont insérées à l'ancienne position de la mère.

Let us remark that delaying by τ the starting time of the tasks of the schedule given by $\mathcal{A}_{C_{max}}$ increases the completion time of the tasks with the same delay τ. The starting point of the new algorithm is the schedule built by $\mathcal{A}_{\sum C_i}$. The tasks ending in this schedule before $\rho_{C_{max}} C^*_{max}$ are left unchanged. All tasks ending after $\rho_{C_{max}} C^*_{max}$ are removed and rescheduled with $\mathcal{A}_{C_{max}}$, starting at $\rho_{C_{max}} C^*_{max}$. As $\mathcal{A}_{C_{max}}$ is able to schedule all tasks in $\rho_{C_{max}} C^*_{max}$ and it is always possible to remove tasks from a schedule without increasing its completion time, all these tasks will complete before $2\rho_{C_{max}} C^*_{max}$.

Now let us look at the new values of the two criteria. Any task scheduled by $\mathcal{A}_{\sum C_i}$ ending after $\rho_{C_{max}} C^*_{max}$ does not increase its completion time by a factor more than 2, thus the new performance ratio is no more than twice $\rho_{\sum C_i}$. On the other hand, the makespan is lower than $2\rho_{C_{max}} C^*_{max}$. Thus the performance ratios on the two criteria are the double of the performance ratio of each single algorithm.
We can also remark that the schedule presented has a lot of idle times and the makespan can be greatly improved by just starting every task as soon as possible with the same allocation and order. However, even if this trick can give very good results for practical problems, it does not improve the theoretical bounds proven on the schedules, as it cannot always be precisely defined.

Tuning performance ratios It is possible to decrease one performance ratio at the expense of the other. The point is to choose a border proportionally to $\rho_{C_{max}} C^*_{max}$, namely $\lambda * \rho_{C_{max}} C^*_{max}$. The performance ratios are a Pareto curve of λ.

Proposition. It is possible to combine $\mathcal{A}_{\sum C_i}$ and $\mathcal{A}_{C_{max}}$ in a new algorithm with a performance ratio of $\frac{1+\lambda}{\lambda}\rho_{\sum C_i}$ and $(1+\lambda)\rho_{C_{max}}$ at the same time.

Combining two existing algorithms it is possible to schedule independent moldable tasks with a performance ratio of 3 for the makespan and 16 for the sum of the completion time.

The former approach required the use of two algorithms, one per criterion and mixed them in order to design a bi-criterion scheduling algorithm. It is also possible to design an efficient bi-criterion algorithm just by adapting an algorithm $\mathcal{A}_{C_{max}}$ designed for the makespan criterion.
The main idea is to create a schedule which has a performance ratio on the sum of completion times based on the result of algorithm $\mathcal{A}_{C_{max}}$ without losing too much on the makespan. To have this performance ratio $\rho_{\sum C_i}$ on the sum of the completion times, we actually try to have the same performance ratio $\rho_{\sum C_i}$ on all

the completion times.

3 Le problème du voyageur de commerce symétrique et la programmation linéaire

3.1 Le problème

Etant donnés un certain nombre de villes et les distances inter-villes, le célèbre problème du Voyageur de Commerce consiste pour un voyageur à visiter toutes les villes exactement une fois et revenir à son point de départ en parcourant la plus petite distance possible. Les distances sont symétriques, elles sont les mêmes dans chaque sens de parcours entre deux villes données.
On modélise le problème par un graphe complet avec des longueurs associées aux arêtes. On supposera le graphe complet, c'est à dire que toutes les liaisons existent. Si ce n'est pas le cas on peut rajouter les liaisons manquantes et mettre sur toute arête une longueur égale à la plus courte distance entre ses extrémités dans le problème original. Remarquez que dans ce cas une solution du nouveau problème ne correspond pas nécessairement à un parcours dans lequel chaque ville n'est visitée qu'une seule fois.
Un cycle est dit *hamiltonien* s'il passe par chaque sommet exactement une fois. Le problème est donc de trouver un cycle hamiltonien de longueur minimum.
Nous utiliserons les notations suivantes :
$K_n = (V, E)$, est un graphe complet avec $n = |V|$ sommets. Soit $S \subset V$, alors $\delta(S)$ (resp. $\gamma(S)$) représente l'ensemble des arêtes avec exactement une extrémité dans S (resp. les deux extrémité dans S), i.e. $\delta(S) = \{(u, v) \in E : u \in S, v \notin S\}$ (resp.$\gamma(S) = \{(u, v) \in E : u \in S, v \in S\}$). On notera $\delta(v)$ au lieu de $\delta(\{v\})$ pour $v \in V$.
On notera $\mathbb{R}^E$ l'ensemble des vecteurs indexés par E, c.à.d. que les composantes d'un vecteur de $\mathbb{R}^E$ sont en bijection avec les éléments de E. Pour $E^\star \subset E$ et $x \in \mathbb{R}^E$, alors $x(E^\star)$ représente $\sum_{e \in E^\star} x_e$. Soit $x^\star \in \mathbb{R}^E$, avec $x^\star_e$ vu comme une *capacité* de l'arête e ; pour $S \subset V$ on appelle $x^\star(\delta(S))$, la *valeur* de la coupe définie par S.

3.2 Le problème du voyageur de Commerce vu comme un programme linéaire en nombres entiers

Il y a plusieurs façons de modéliser le problème du voyageur de commerce comme un programme linéaire en nombres entiers. Une seule a été pour le moment utilisée comme base d'un processus de résolution, celle connue sous le nom de *formula-*

tion à deux indices. A chaque arête $e \in E$ on associe une variable x_e qui vaudra 1 si e est dans la solution optimale et 0 sinon. Le problème du voyageur de Commerce peut alors se formuler comme le programme linéaire en nombres entiers suivant :

$$\text{(IP(STSP))} \qquad \min \sum_{e \in E} c_e x_e \tag{5}$$

subject to

$$x(\delta(v)) = 2 \quad \text{for} \quad v \in V \tag{6}$$

$$x(\delta(S)) \geq 2 \quad \text{for} \quad 3 \leq |S| \leq |V|/2 \tag{7}$$

$$0 \leq x_e \leq 1 \quad \text{for} \quad e \in E \tag{8}$$

$$x_e \text{ integer} \quad \text{for} \quad e \in E \tag{9}$$

Equations (6) disent qu'exactement deux arêtes sont incidentes à un sommet. Inequalities (7) empêchent la création de cycles qui ne contiendraient pas tous les sommets et sont par conséquent appelées *inéquations d'élimination de sous tours*. Notez que le nombre de ces inéquations est exponentiel et interdit donc la résolution du programme linéaire obtenu en relâchant les contraintes de valeurs entières sur les variables, par des méthodes classiques. Cependant ce programme linéaire ne pose pas trop de problèmes du fait que trouver une inéquation de sous-tour non vérifiée est un problème bien résolu, à savoir un problème de coupe de capacité minimum.
On répète le processus suivant : On part du programme linéaire sans les contraintes de sous tour, on résout par un logiciel de programmation linéaire, on obtient une solution x^*, en utilisant les valeurs x_e^* comme des capacités on résout le problème de la coupe minimum, si celle-ci est de valeur 2, aucune inéquation de sous tour n'est violée, sinon on a trouvé une telle inéquation violée, on la rajoute au programme linéaire et on itère jusqu'à ne plus trouver de telle inéquation. On n'a aucune garantie de terminer rapidement, cependant par expérience, c'est le cas pour toutes les instances connues.

3.3 Résolution du programme linéaire en nombres entiers

A partir de là on pourrait entrer dans une méthode classique de Branch and Bound. Malheureusement cette méthode ne donne aucun résultat même pour des instances de taille modeste.
Il faut renforcer la formulation. On sait que n'importe quel programme linéaire en nombres entiers peut se transformer en programme linéaire par l'ajout d'un nombre fini de contraintes. Pour le Problème du voyageur de commerce il est peu probable qu'un jour on connaisse l'ensemble de ces contraintes. On ne les connaît que jusqu'à 10 villes. Pour donner une idée de la complexité voici pour

n	# tours	# contraintes différentes	# classes
3	1	0	0
4	3	3	1
5	12	20	2
6	60	100	4
7	360	3 437	6
8	2 520	194 187	24
9	20 160	42 104 442	192
10	181 440	$\geq$ 51 043 900 866	$\geq$ 15 379

TAB. 1 – Statistiques

les premières valeur de $n = |V|$ le nombre minimum de contraintes nécessaires. Dans le tableau suivant deux inéquations sont dans la même classe si les coefficient s'obtiennent par renumérotation des sommets. Ayez en tête que l'on résout relativement facilement aujourd'hui des instances de 2000 villes.
On donnera quelque exemples d'inéquations permettant de renforcer la formulation.

Références

[1] Guy E. Blelloch, Phillip B. Gibbons, Yossi Matias, and Girija J. Narlikar. Space efficient scheduling of Parallelism with Synchronization Variables. In *Proceedings of the 9th Symposium on Parallel Algorithms and Architectures*. ACM Press, June 1997.

[2] A. Borodin. On relating time and space to size and depth. *SIAM Journal of Computing*, 6 :733–744, 1977.

[3] A. Borodin and I. Munro. *The Computational Complexity of Algebraic and Numeric Problems* . Elsevier, New-York, 1975.

[4] R.P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21 :201–206, 1974.

[5] E.G. Coffman and P.J. Denning. *Operating Systems Theory*. Prentice-Hall, 1973.

[6] S.A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64 :2–22, 1985.

[7] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The Implementation of the Cilk-5 Multithreaded Language. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'98)*, June 1998.

[8] J. von zur Gathen. Parallel arithmetic computations : a survey. In *Proc. 12th Int. Symp. Math. Found. Comput. Sci., Bratislava*, pages 93–112. Springer-Verlag LNCS 233, 1986.

[9] A.M. Gibbons and W. Rytter. *Efficient parallel algorithms*. Cambridge University Press, 1988.

[10] L.M. Goldschlager. The monoton and planar circuit value problems are log space complete for P. *SIGACT News*, 9 :25–29, 1977.

[11] R.L. Graham. Bounds for Certain Multiprocessor Anomalies. *Bell System Tech J.*, 45 :1563–1581, 1966.

[12] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2) :416–426, 1969.

[13] H.J. Hoover, M.M. Klawe, and N.J. Pippenger. Bounding fan-out in logical networks. *J. ACM*, 31 :13–18, 1984.

[14] E. Kaltofen, G.L. Miller, and V. Ramachandran. Efficient parallel evaluation of straight-line code and arithmetic circuits. *SIAM J. Computing*, 17 :687–695, 1988.

[15] R.M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leuwen, editor, *Algorithms and Complexity*, pages 869–932. Elsevier, 1990.

[16] Aicha Kerfali, Jean-Louis Roch, and El Mostafa Daoudi. Algorithmes parallèles à grain adaptatif - Application à la parallélisation de gzip. In *RENPAR'15*, pages 18–26, Nice, France, octobre 2003.

[17] Jean-Claude Konig and Jean-Louis Roch. Machines virtuelles et techniques d'ordonnancement. In *Proceedings école d'hiver de PRS ICARE'97*, Auusois, France, dec 1997. http ://www-apache.imag.fr/ jlroch/ps/97-sched-aussois.ps.gz.

[18] Eric Mohr, David A. Kranz, and Jr Robert H. Halstead. Lazy task creation : A technique for increasing the granularity of parallel programs. *IEEE Transactions on Parallel and Distributed Systems*, 2(3) :263–280, July 1991.

[19] Girija J. Narlikar and Guy E. Blelloch. Space-efficient implementation of nested parallelism. In *Proceedings of the Sixth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 25–36, June 1997.

[20] Victor Y. Pan and Franco P. Preparata. Work-preserving speed-up of parallel matrix computations. *SIAM Journal on Computing*, 24(4), 1995.

[21] N. Pippenger. On simultaneous ressource bounds. In *20 th. Annual IEEE Foundations of Computer Science*, pages 307–311, 1979.

[22] N. Revol. Evaluation parallèle de circuits arithmétiques. Technical Report Pré-rapport de thèse, IMAG, 1993.

[23] J.-L. Roch, T. Gautier, and R. Revire. Athapascan : Api for asynchronous parallel programming. Technical Report RT-0276, INRIA Rhône-Alpes, projet APACHE, February 2003.

[24] W.L. Ruzzo. On uniform circuit complexity. *J. Computer and System Sciences*, 22, 3 :365–383, 1981.

[25] D. B. Shmoys, J. Wein, and P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6) :1313–1331, 1995.

Algorithmes pour l'image de synthèse

Responsable : Gilles Debunne, CNRS.

Conférenciers : Xavier Décoret, Cyril Soler

Sommaire

Algorithmes pour l'image de synthèse

Gilles Debunne, Xavier Décoret, Cyril Soler[4]

Projet ARTIS[5]

1 Introduction

L'image de synthèse est un domaine très jeune (une trentaine d'années tout au plus) et qui vit en ce moment une véritable révolution technologique et scientifique. Cette discipline combine une forte composante mathématique (en particulier géométrique) et algorithmique (liée à la taille des données traitées).
Ce cours se propose de présenter quelques uns des formalismes mathématiques spécifiques au domaine de l'informatique graphique, ainsi que certaines méthodes de résolution originales. Même si les thèmes abordés sont éloignés de ceux des participants, les mathématiques (principalement basées sur l'algèbre linéaire) et les algorithmes (dédiés aux données volumineuses et souvent hiérarchiques) qui sont utilisés doivent pouvoir être rapprochés de ceux couramment utilisés par tout scientifique. Les auteurs espèrent ainsi que cette présentation ouverte pourra donner lieu à des recoupements, à des généralisations, voire à de nouveaux thèmes d'application ou de nouvelles pistes.

1.1 Présentation du domaine

Avec l'évolution de la taille des disques durs, les cartes graphiques sont le seul domaine de l'informatique à *dépasser* les prévisions de la loi de Moore, qui prévoit le doublement de la puissance tous les 18 mois. Ce décuplement de puissance se répercute très rapidement dans les machines grand public, tiré par le développement de l'industrie du jeu vidéo, qui génère désormais un chiffre d'affaire supérieur à celui de l'industrie cinématographique.
Il en résulte un grand chamboulement de la discipline, une effervécence liée aux nouvelles possibilités offertes, et à celles que la recherche peut proposer, qui peuvent être intégrées à très court terme dans la prochaine génération de cartes. Cette évolution ne saurait rendre caduque les problématiques liées à la complexité des scènes, bien au contraire, la demande pour des univers plus riches et plus beaux augmentant sans doute plus rapidement que les performances brutes des cartes.

[4]Adresses électroniques : `prenom.nom@imag.fr`
[5]`artis.imag.fr`

Cette évolution va également très prochainement déborder sur d'autres domaines. Une carte graphique peut en effet être considérée comme un co-processeur mathématique parallèle très performant (supérieur à tout Pentium) dédié au calcul matriciel $4*4$, qui est plus est de bas prix grâce aux économies d'échelle. Son utilisation dans la résolution de problèmes mathématiques a d'ores et déja était présentée. De même les interfaces graphiques pourront-elles tirer parti d'un affichage en 3D pour proposer de nouvelles méthodes d'interaction.

1.2 Objectifs du cours

L'introduction faite dans ce cours devrait familiariser le lecteur avec le domaine de la synthèse d'images. Celui-ci est devenu très abordable et nous espérons vous convaincre de franchir le pas. La visualisation tri-dimentionelle peut en effet être un outil puissant de compréhension et d'analyse de données complexes, éventuellement grâce à leur animation, et permet également d'observer de façon plus intuitive les résultats d'un algorithme. Les bases données dans ce cours ainsi que les nombreux exemples de code à copier-coller référencés devraient vous permettre de rapidement utiliser cet outil.
L'informatique graphique est également un excellent domaine d'application pour de très nombreuses recherches algorithmiques ou mathématiques. La liste en est très longue, mais on peut citer la résolution de gros systèmes linéaires, la compression de données, le traitement du signal que représentent une image ou un modèle 3D, les algorithmes probabilistes, l'intégration numérique, la résolution d'équations aux dérivées partielles, et bien d'autres encore.
Comme dit en introduction, les auteurs espèrent créer des rapprochements entre les diverses disciplines des participants et les notions présentées dans ce cours. cette ouverture d'esprit à d'autres disciplines ne peut en tout cas qu'être bénéfique et nous l'espérons agréable voire fructueuse.
Les notions présentées dans ce cours sont issues d'une sélection arbitraire, où l'on a chercher à faire une introduction, à couvrir des thèmes bien différents en espérant toucher un large public, à mélanger les aspects mathématiques et algorithmiques, tout en restant simple et facilement accessible. Ceci explique l'hétérogénéïté de ce cours, ainsi que la simplification parfois effectuée, au détriment de la complétude de l'explication. Le lecteur intéressé est référé à la bibliographie ou à une discussion plus approfondie avec les intervenants.

2 Résumés des cours

Nous ne résumerons ici que très succintement les cours, afin de les rendre plus compréhensibles au lecteur qui ne pourra y participer. Les transparents se veulent

suffisemment détaillés pour en permettre une lecture autonome.

2.1 Les coordonnées homogènes

Ce formalisme mathématique consiste à plonger l'espace tri-dimentionnel couramment utilisé en image de synthèse dans un espace de dimension 4. Le bénéfice principal est une uniformisation sous forme de produit matrice-vecteur de toutes les opérations classiques appliquées en 3D : rotation, changement d'échelle et surtout translation, ce qui aurait été impossible sans les coordonnées homogènes. On y gagne de plus une représentation élégante de la notion de point à l'infini qui s'intègre parfaitement avec le cas général dans les applications de géométrie projective. Ce formalisme est ainsi adopté dans toutes les disciplines utilisant des coordonnées 3D : imagerie, vision et robotique.

2.2 Création d'une image de synthèse

Nous décrivons ensuite les bases de la création d'une image, à savoir la projection sur un plan image d'une géométrie 3D. Grâce aux coordonnées homogènes, cette projection peut elle aussi être exprimée par une matrice $4 * 4$, que l'on applique aux points 3D pour en obtenir la projection. Il faudrait pour compléter cette présentation parler du calcul de la couleur des points projetés, en introduisant les notions de *modèles de matériaux*, qui modélisent la relation entre les couleurs (au pluriel) de la surface, les couleurs des lampes, leurs positions respectives et la normale à la surface.
On donne ensuite quelques éléments sur l'organisation logicielle et matérielle liée à l'image de synthèse, ainsi que sur son évolution. Celle-ci est devenue beaucoup plus modulaire et les cartes graphiques sont désormais programmables.

2.3 Méthode de radiosité

Le but est ici de calculer l'éclairage reçu par les objets d'une scène, en présence de sources lumineuse. On montre qu'après simplification des hypothèses et discrétisation en éléments finis, le problème peut se ramener à la résolution d'un système matriciel. Chaque élément du vecteur solution correspond à la radiosité sur un petit élément de surface, l'ensemble de ces éléments formant les surfaces des objets de la scène. Les méthodes classiques se révèlent inefficaces, car trop sensibles aux erreurs numériques ou trop lentes.
La méthode employée consiste alors en une résolution *itérative* du problème, par applications successives de mêmes opérations matricielles. On montre la validité mathématique de cette méthode et en présente une interprétation physique. Chaque itération revient à transférer de l'énergie lumineuse entre l'ensemble des

éléments de la scène. L'une des méthodes de résolution utilisées (le *shooting*) revient à propager de l'énergie depuis les sources sur les éléments, qui eux-mêmes la renvoie aux autres éléments, et ainsi de suite. L'autre méthode (le *gather* est duale : chaque élément somme l'ensemble de l'énergie reçue après un nombre donné de refléxions.
Bien que convergent assez rapidement, cet algorithme est trop lent pour des scènes complexes et nous présentons sa version hiérarchique, qui a démocratisé son utilisation (même si celle-ci reste limitée dans la pratique, le *lancer de rayon* étant généralement préféré).

2.4 Les quaternions

Les quaternions sont LA représentation adaptée des rotations dans $\mathcal{R}^3$. Après avoir présenté plusieurs alternatives, toutes ayant leurs défaut, nous montrons que les quaternions, qui ne sont rien d'autre que des points sur la sphère unité de $\mathcal{R}^4$, s'adaptent parfaitement à la représentation des rotations. L'algèbre des quaternions est très propre et permet en particulier de très naturellement de composer deux rotations, d'appliquer une rotation à un point ou encore et surtout d'interpoler entre deux rotations.

2.5 Algorithmes de simplication de maillages

Les maillages 3D utilisés pour représenter des objets s'avèrent inutilement complexes lorque l'objet est vu de loin et donc affiché en petit. Le niveau de détail permet alors de générer plusieurs représentations d'un même objet, chacune étant adaptée à un intervalle de taille d'affichage. Nous présentons ici les différentes classes d'algorithmes qui permettent de décimer un maillage 3D, en concervant ou non certaines propriétés utiles (topologie, silhouette, apparence générale...). Nous présentons enfin dans le détail l'algorithme de Garland, très utilisé en pratique et (car) disponible gratuitement. Son formalisme mathématique simple, efficace et astucieux ainsi que ses très bons résultats pratiques (sur de gros modèles) en font une méthode de choix. Il est détaillé dans la suite de ces notes de cours.

3 Références

La référence reconnue du domaine est la conférence *Siggraph*, dont les actes sont maintenant publiés dans un numéro spécifique de la revue TOG (*Transactions on Graphics*), publiée par l'ACM. Nous reportons donc dans un premier temps le lecteur intérressé par les développements majeurs de la discipline aux articles de cette revue.

Le livre de référence en image de synthèse est le *Computer Graphics* de Foley, van Dam, Feiner et Hughes, publié chez Addison-Wesley. Une nouvelle édition remise à jour est prévue pour 2004.
Cette référence ainsi que celles complétant le cours sont regroupées dans les dernières diapositives du cours. Les auteurs sont bien-sûr également joignables par courriel pour des questions plus précises.

Quadric Simplification en trois mots

Xavier Décoret

1 Présentation

Quadric Simplification est un algorithme de *simplification de maillage* mise au point par Michael Garland et Paul Heckbert en 1997 [GH97]. Cette approche utilise un opérateur local d'*effondrement de paires de points* et une métrique d'erreur *sommet-plans*. Pour une présentation des différents opérateurs de simplifications de maillage et des métriques d'erreurs, on consultera le livre de Luebke et al., chap. 2 et 3 [LRC+02]. Un maillage 3D est simplifié en remplaçant deux points par un seul, comme illustré sur la figure 3. Une fois cette opération effectuée, on supprime les faces dégénérées et on met à jour les relations d'adjacence.

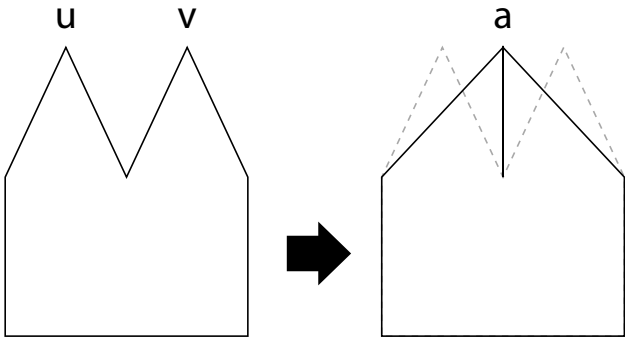


FIG. 3 – Simplification par effondrement de paires de points

L'erreur commise est définie sur les sommets du maillage initial, en mesurant de combien ils s'éloignent des plans supports des faces auquels ils appartiennent. Un sommet u qui, après une séquence d'effondrements le concernant, se retrouve au point a, entraîne donc une erreur :

$$e(u \rightarrow a) \equiv \sum_{F \in \lceil\lceil u \rceil\rceil} d^2(a, \mathcal{P}(F)) \tag{10}$$

où les F sont les faces adjacentes à u. La figure 4 illustre cette définition. Le carré de la distance d'un point $(x, y, z)^T$ à un plan est une forme quadratique en x, y, z et peut s'écrire, en coordonnées homogènes :

$$d^2(a, \mathcal{P}(F)) = a^T Q_F a \tag{11}$$

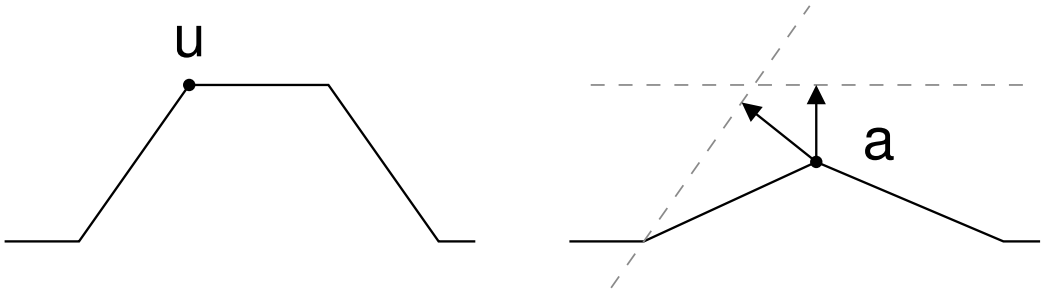


FIG. 4 – Erreur pour un sommet. Si u se retrouve en a, l'erreur est la somme du carré des distances aux plans en pointillés.

avec $a = (x, y, z, 1)^T$ et où Q_F est une matrice 4×4[6]. En injectant dans (10) et par linéarité, on obtient :

$$e(u \to a) = a^T Q_u a \tag{12}$$

où $Q_u = \sum_{F \in \lceil\lceil u \rceil\rceil} Q_F$ est dite *quadrique d'erreur* de u. On a triviallement $u^T Q_u u = 0$ car u appartient à toutes les faces adjacentes !

2 Intérêt des quadriques d'erreurs

2.1 Compacité mémoire

Supposons que l'on effectue les effondrements $u, v \mapsto a$ puis $a, w \mapsto b$ comme indiqué sur la figure 5. Pour mesurer l'erreur commise lors de la dernière opération,

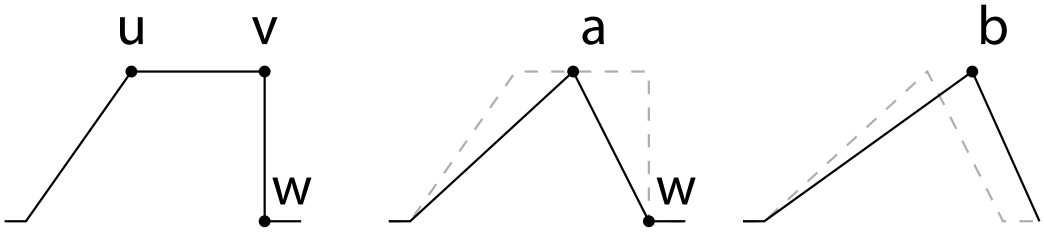


FIG. 5 – Succession d'effondrements

il faut se rappeller que a a été obtenu auparavant par effondrement de u et v. Maintenir cette information pour tous les points introduits lors de la simplification est

[6]Si le plan est représenté par $p = [abcd]^T$ alors $Q = pp^T$

prohibitif et inefficace. Heureusement, c'est inutile ! En effet :

$$\begin{aligned} e(u \to b) + e(v \to b) &= b^T Q_u b + b^T Q_v b \\ &= b^T (Q_u + Q_v) b \\ &= e(a \to b) \\ &\quad \text{avec } Q_a \equiv Q_u + Q_v \end{aligned}$$

Lorsque l'on crée un nouveau point a, il suffit donc de lui associer comme quadrique d'erreur la somme des quadriques des deux points qu'il remplace. Cette quadrique garde implicitement trace des plans auquels appartenaient tous les sommets qui se "retrouvent" dans a !

2.2 Optimisation

Lorsque l'on décide d'effondrer deux sommets u et v, il faut choisir par quel point les remplacer. Idéalement, on voudrait prendre celui qui minimise l'erreur commise. Garland et Heckbert ont montré que ce point est donné par :

$$a = \frac{1}{2} \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 2 \end{bmatrix} \tag{13}$$

lorsque la matrice est inversible. Dans le cas contraire, ils proposent de calculer l'erreur en u, en v et au milieu de $[u, v]$ et de choisir celui qui minimise l'erreur.

3 Algorithme

L'algorithme complet est le suivant. On initialise les quadriques de chaque sommets du modèle initial. On considère toutes les paires de sommets adjacents (reliés par une arrête) ou proches géométriquement[7] Pour chacune d'entre elles, on calcule le meilleur point où les effondrer et l'erreur (minimisée) correspondante. On insère les paires dans une liste de priorité classée par erreur croissante. On effondre la paire sur le dessus de la liste. On met à jour les nouvelles paires et on réitère jusqu'à obtenir un nombre de faces fixé, où une erreur maximale autorisée.

[7] C'est la partie "sensible" de l'algorithme. Considérer toutes les paires entraîne une complexité quadratique. Il faut un peu de doigté pour choisir quelles paires considérer.

4 Conclusion

Cet algorithme est très efficace en pratique. Il a été étendu [GH98] pour tenir compte des attributs spécifiés à la surface du maillage (normale, couleur, coordonnées textures...). Une implémentation est disponible sur le web (`http://graphics.cs.uiuc.edu/~garland/software/qslim.html`).

Références

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. pages 209–216, 1997.

[GH98] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. pages 263–269. IEEE-CS Press, 1998.

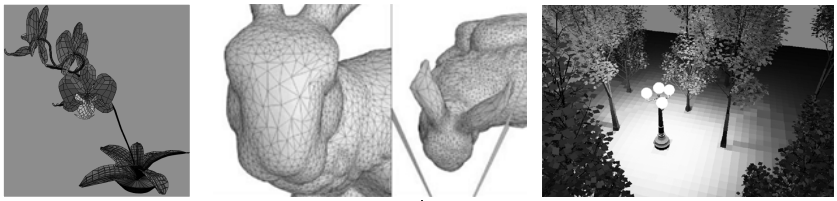
[LRC+02] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, first edition, july 2002.

Algorithmes pour l'image de synthèse

Gilles Debunne

Xavier Décoret

Cyril Soler



Introduction

- Domaine de l'image de synthèse très actif
- Combine mathématiques et algorithmique

- Présentation d'outils et de formalismes
  - Coordonnées homogènes
  - Quaternions
- Détail de deux algorithmes
  - Radiosité
  - Simplification de maillage

Plan du cours

- Coordonnées homogènes
- Création d'une image de synthèse
- Calculs de radiosité

Pause

- Quaternions pour les rotations
- Algorithmes de simplication de maillages

Les coordonnées homogènes

Un formalisme unifié et puissant pour représenter les transformations géométriques

Transformations géométriques

- On manipule des points en 3D
- On doit leur faire subir des transformations

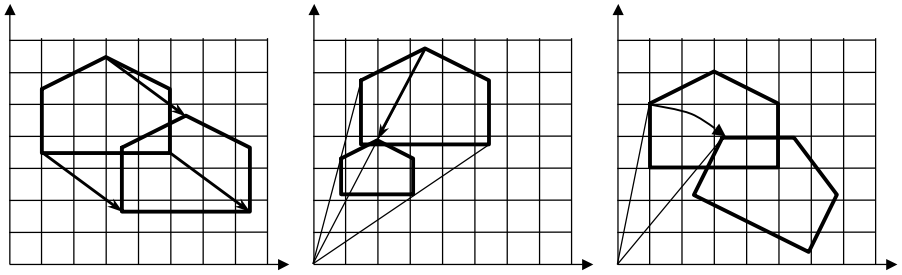
  Translation, rotation, changement d'échelle

  $\rightarrow$Trouver un formalisme unifié

- On représente les points par des vecteurs

$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Représentation matricielle



$P' = P + T$ $\quad P' = S \bullet P$ $\quad P' = R \bullet P$

$$T = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \qquad S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix} \qquad R = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(selon axe Z)

Coordonnées homogènes

- Représentation non unifiée

  Addition vectorielle ou multiplication matricielle ?

- Solution : plonger dans $\mathbb{R}^4$

$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \approx \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \qquad \begin{pmatrix} xw \\ yw \\ zw \\ w \end{pmatrix} \approx \begin{pmatrix} x / w \\ y / w \\ z / w \end{pmatrix}$$

- Bonus

w=0 permet de représenter un point situé à l'infini

Modifications

- Pour les rotations et les chgt d'échelle

  Il suffit de compléter la matrice

$$R=\begin{pmatrix} \boxed{R} & \begin{matrix}0\\0\\0\end{matrix} \\ 0\ 0\ 0 & 1 \end{pmatrix} \qquad S=\begin{pmatrix} \boxed{S} & \begin{matrix}0\\0\\0\end{matrix} \\ 0\ 0\ 0 & 1 \end{pmatrix}$$

$$P' = M \bullet P = \begin{pmatrix} \boxed{M} & \begin{matrix}0\\0\\0\end{matrix} \\ 0\ 0\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} M \times P \\ \\ w \end{pmatrix} \approx M \bullet P$$

Translation par multiplication

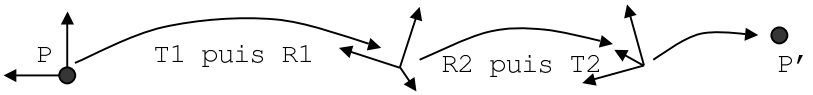
- On définit ainsi la matrice de translation

$$T=\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P' = T \bullet P = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x + w\,t_x \\ y + w\,t_y \\ z + w\,t_z \\ w \end{pmatrix} = P+T$$

Composition de transformations

- Il suffit de multiplier les matrices associées

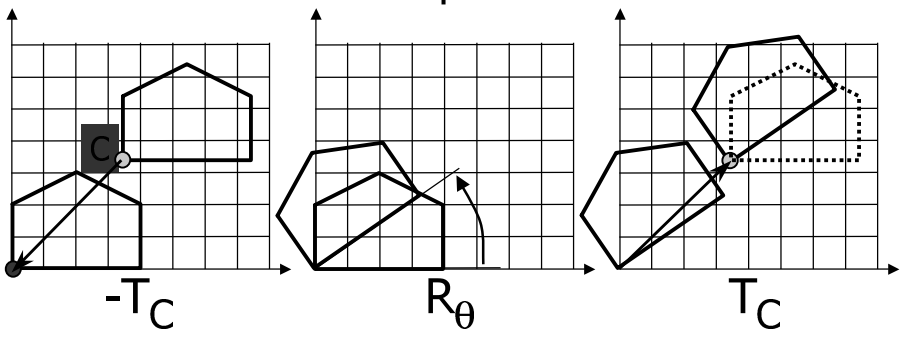


$$P' = T_2 \bullet R_2 \bullet R_1 \bullet T_1 \bullet P = M \bullet P$$

- L'ordre des multiplications est important
- Toute transformation peut ainsi être représentée par une matrice 4x4

Exemple : rotation excentrée

- Rotation autour d'un point C arbitraire



$$R_{\theta,C} = T_C \bullet R_\theta \bullet T_{-C}$$

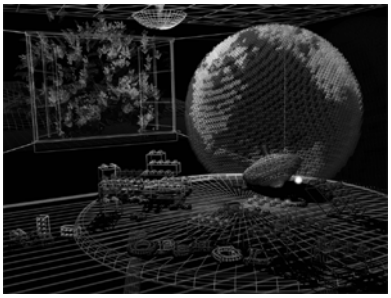
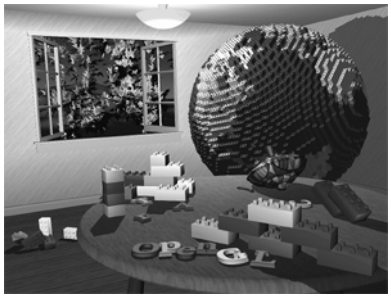
Applications des coord. homog.

- Unification (tout est multiplication)
- Généralisation (notion de point à l'infini)

- Enomément utilisées
  En vision, robotique, géométrie projective
- Représentent les systèmes de coordonnées
  Une transformation est un changement de repère

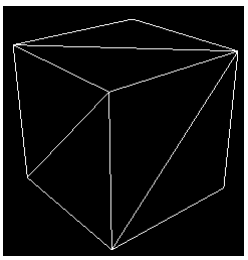
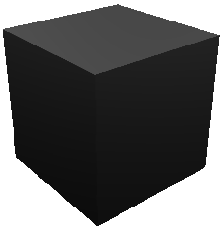
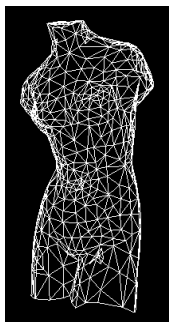
Création d'une image de synthèse

Principes et
calcul de la projection

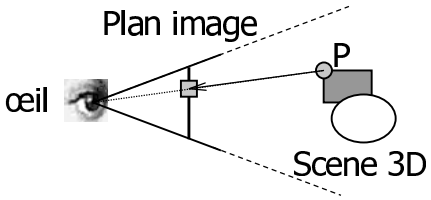
Description de la scène 3D

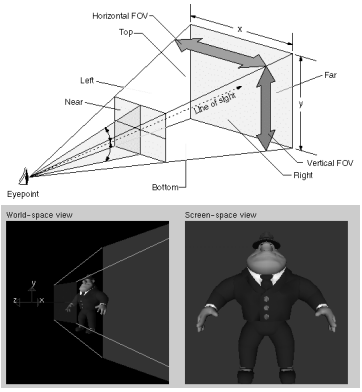
- Un ensemble d'objets
  - La surface de ces objets
    - Un ensemble de triangles
      - Les positions de chaque sommet (x,y,z)
      - Les couleurs, normales, … de ces sommets


Description de la caméra

- Position et orientation dans la scène
- Généralement de type projectif
  Points projetés dans la direction de l'œil

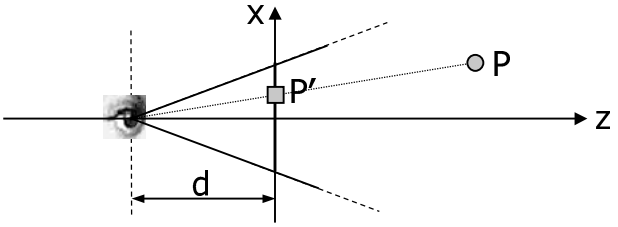




En 3D : pyramide de vue

Projection 3D → 2D

- Dans cette configuration
  Direction de vue alignée avec l'axe z
  Axes de l'écran alignés sur x et y
  Plan image en z=0, œil en z = -d (distance focale)



- On a (Thalès) : $P_x / (P_z+d) = P'_x / d$

Représentation matricielle

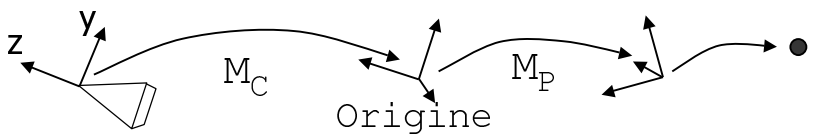
$$P'_x = d\,P_x / (P_z + d)$$

$$P' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d + w \end{pmatrix} \approx \begin{pmatrix} dx/(z+dw) \\ dy/(z+dw) \\ dz/(z+dw) \\ 1 \end{pmatrix}$$

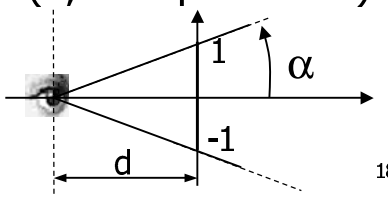
- Division par P_z rendue possible par les coordonnées homogènes

Positionnement arbitraire

- Lorsque la caméra n'est pas alignée selon z

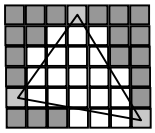


- On se ramène au cas précédent en • par M_C
- En pratique d = 1 / tan α (α, champ de vision)
- Coordonnées écran ramenées dans [-1, 1]



Organisation logicielle

- CPU envoie les ordres graphiques au GPU
- GPU : co-processeur dédié au calcul 4x4
  - Projection des points, *rasterization*
  - Elimination des parties cachées, z-buffer
  - Plaquage de textures sur les polygones
  - Cartes programmables



- Exemples de code à copier-coller sur

`artis.imag.fr/~Gilles.Debunne/Enseignement/3DAPI/`

Application interactive

- Squelette classique : boucle infinie
  - Interaction avec l'utilisateur
  - Calculs, déplacement de la caméra
  - Affichage de la scène 3D
    - Effacer l'écran
    - Description de la caméra
    - Description de la scène

`artis.imag.fr/~Gilles.Debunne/QGLViewer`

Evolution de la discipline

- Explosion du marché et du matériel
- Loi de Moore dépassée, révolution en cours
- Tendances
  - De plus en plus de détails (~400M triangles/sec)
  - Plus de géométrie, maléable : *vertex shader*
  - Géométrie plus petite que le pixel : *pixel shader*

Calcul d'éclairage global
Méthode de radiosité

Une méthode originale d'inversion de matrice

Contexte

- Une scène virtuelle
  - Un ensemble d'objets, définis par leurs surfaces
  - Un ensemble de sources lumineuses
- Quelle lumière reçoit chaque surface ?

- Eclairage contribue beaucoup au réalisme
- Simulation "précise" souhaitée (architecture)

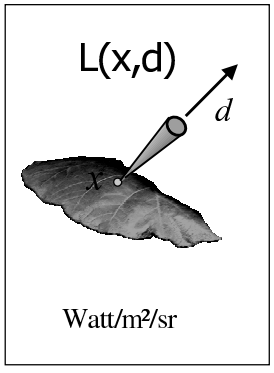
Hypothèses et principes

- Pas de phénomène ondulatoire de la lumière
- Etablissement d'un régime permanent
- Par d'interaction avec l'air (brouillard, …)

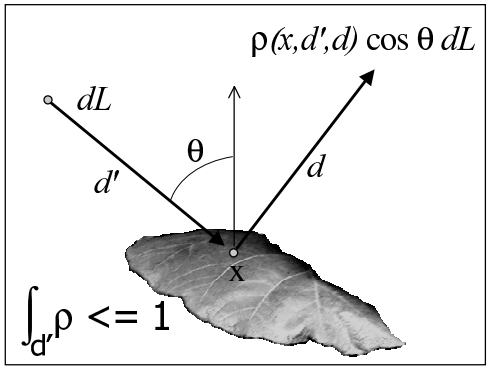
- La lumière est émise par les sources
- Elle se propage en ligne droite
- Elle est réfléchie et absorbée par les surfaces

Energie lumineuse

Radiance L



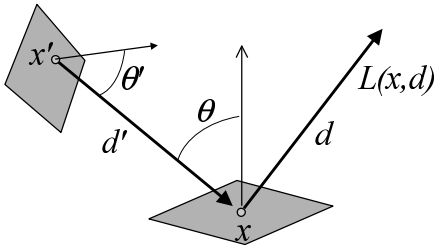
Réflectance ρ

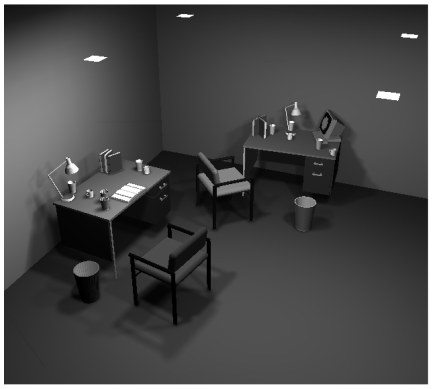


cos(θ) dû à la projection

Equation d'équilibre de la radiance

$$\underbrace{L(x,d)}_{\text{Radiance}} = \underbrace{E(x,d)}_{\text{Emittance}} + \int_{x'} \underbrace{\rho\,(x,d,d')}_{\text{Réflectance}}\ \underbrace{v(x,x')}_{\text{Visibilité}}\ cos(\theta)\ \underbrace{dL(x',d')}_{\text{Angle solide}}$$

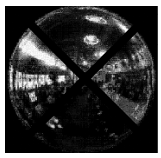




Pas de solution analityque

Hypothèse diffuse

- On suppose indépendants de la direction :
  - La réflectance de toutes les surfaces
  - L'émittance sur les sources



- On pose $B(x) = \int_{\Omega} L(x,d)\ cos(\theta)\ d\omega = \pi L(x,.)$ (W/m²)

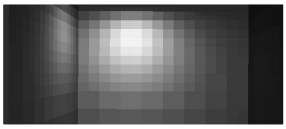
$$\underbrace{B(x)}_{\text{Radiosité}} = \underbrace{\pi E(x)}_{\substack{\text{Emittance}\\ \text{diffuse}}} + \underbrace{\pi\,\rho\,(x)}_{\substack{\text{Réflectance}\\ \text{diffuse}}} \int_{x'} G(x,x')\ B(x')\ dx'$$

- Solution indépendante du point de vue

Radiosité - Plan

- Introduction, notations et hypothèses
- Discrétisation du problème
  - Eléments finis
  - Projection sur une base de fonctions
  - Facteurs de forme
- Résolution de l'équation matricielle
- Méthode hiérarchique

Discrétisation



- $B(x)$ approchée sur une base de fonctions β_i

$$B(x) = \Sigma_i b_i \beta_i + \Delta B = B'(x) + \Delta B$$

- On projète l'équation dans cette base

$$B'(x) = E(x) + \int_{x'} G(x,x') B'(x') dx' + \Delta$$

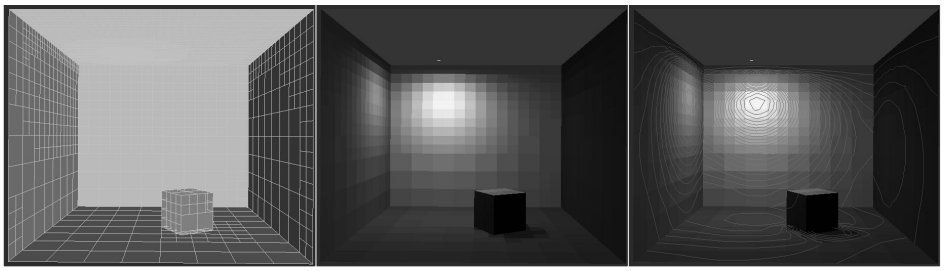
- On essaye d'annuler le résidu Δ
- Méthode de Galerkin : Δ orthogonal aux β_i
- Classiquement : $\beta_i \equiv 1$ sur la facette i, 0 ailleurs
- On cherche $B'(x)$, représenté par un vecteur B

Méthode de Galerkin

- Résolution d'une équation matricielle

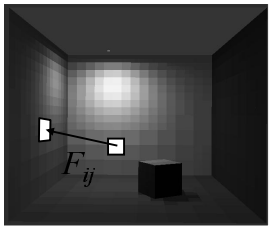
$$B_i = E_i + \rho_i \sum_j F_{ij} B_j$$

- F_{ij} : facteur de forme
- La ligne i représente l'équilibre de la facette i

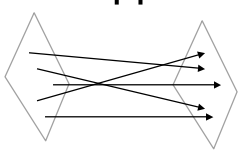


Facteurs de forme F_{ij}

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} v(x,y) \frac{\cos(\theta)\cos(\theta')}{\pi d(x,y)^2} dx\, dy$$



- Proportion d'énergie quittant A_j et atteignant A_i
- Réciprocité $A_i F_{ij} = A_j F_{ji}$
- Conservation de l'énergie $\sum_j F_{ij} \leq 1$
- Calcul complexe, généralement approché
  - Calcul semi-analytique
  - Techniques de Monte-Carlo



Radiosité - Plan

- Introduction, notations et hypothèses
- Discrétisation du problème
- Résolution de l'équation matricielle
  - Méthode spécifique nécessaire
  - Transferts d'énergie itératifs entre surfaces
  - Raffinement progressif du maillage
- Méthode hiérarchique

Résolution de l'équation

$$B_i = E_i + \rho_i \sum_j F_{ij} B_j \qquad (I - M)\, B = E$$

- Espace de spectrum
- Matrice non symétrique et pleine
- Gauss impraticable ! (N > 500)
- QR marche, mais est coûteux

Résolution itérative

- Soit une norme matricielle $S_\varphi(A) = \sup_{x \neq 0} \frac{\varphi(Ax)}{\varphi(x)}$
- Alors $\varphi(A.X) <= S_\varphi(A)\ \varphi(X)$
  $S_\varphi(A \bullet B) <= S_\varphi(A) \bullet S_\varphi(B)$
- S'il $\exists\ S_\varphi$ tq $S_\varphi(M) < 1$, alors $I\text{-}M$ est inversible et
  $(I\text{-}M)^{-1} = \sum_{k=0}^{\infty} M^k \qquad S_\varphi\left((I\text{-}M)^{-1}\right) \leq \frac{1}{1 - S_\varphi(M)}$ [Atkinson'89]
- Or $S_{\varphi_\infty}(M) \leq \rho_{\max} < 1$
- On peut donc résoudre le système en itérant
  $B^{n+1} = E + MB^n \qquad B^0$ indifférent

Conditionnement du problème

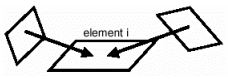
- Soit $\kappa_\varphi(A) = S_\varphi(A) S_\varphi(A^{-1})$ (conditionnement matriciel)
- On a
  $\frac{1}{\kappa_\varphi(A)} = \inf_{\det(B)=0} \frac{S_\varphi(A-B)}{S_\varphi(A)}$
  or
  $1 \leq \kappa_{\varphi_\infty}(I-M) \leq \frac{1+\rho_{\max}}{1-\rho_{\max}}$

Le problème est très bien conditionné (peu sensible aux instabilités) pour des scènes à faible réflectance (<0.5)

- Les itérations de propagation de la lumière convergent d'autant plus vite que ρ_{max} est petit
  $\varphi_\infty(B^n - B) \leq \rho_{\max}^n \varphi_\infty(B^0 - B)$

Deux types de résolution itérative

- Gathering

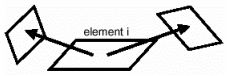


$$B_i^{k+1} = E_i + \rho_i \sum_{j=1}^{i-1} F_{ij} B_j^{k+1} + \rho_i \sum_{j=i+1}^{n} F_{ij} B_j^k$$

ou

$$B_i^{k+1} = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j^k$$

- Shooting

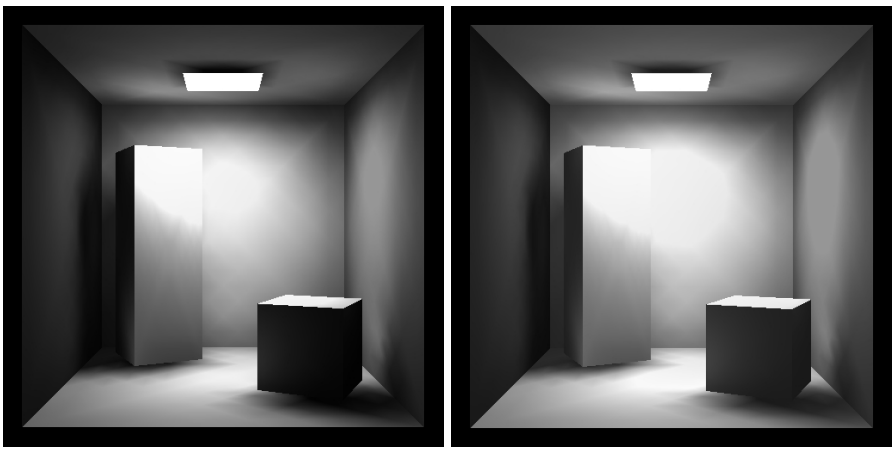


Initialisation $\Delta B_j^{n+1} = 0$

Puis, pour tout i,j $\Delta B_j^{n+1} \mathrel{+}= \Delta B_i^n \rho_j F_{ji}$ $\Delta B_i^0 = E_i$

$B^{n+1} \mathrel{+}= \Delta B^{n+1}$ $B_i^0 = E_i$

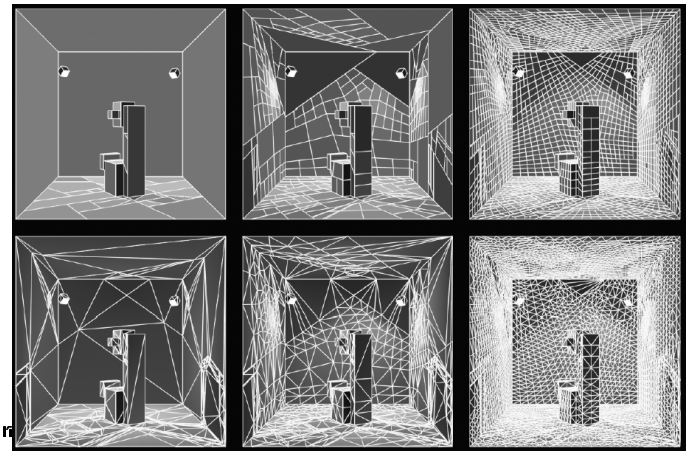
Exemple de résolution itérative



Départ de zéro Départ d'un terme constant

Raffinement progressif

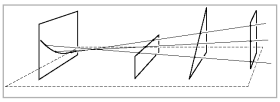
- Maillage créé en fonction de la variation de l'irradiance et des discontinuités



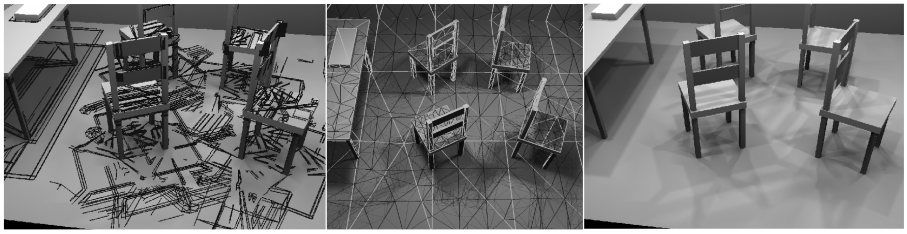
Maillage de discontinuité

- Difficultés

  Discontinuités nombreuses et de nature variable



Evênement triple-arête ⇨ quadrique

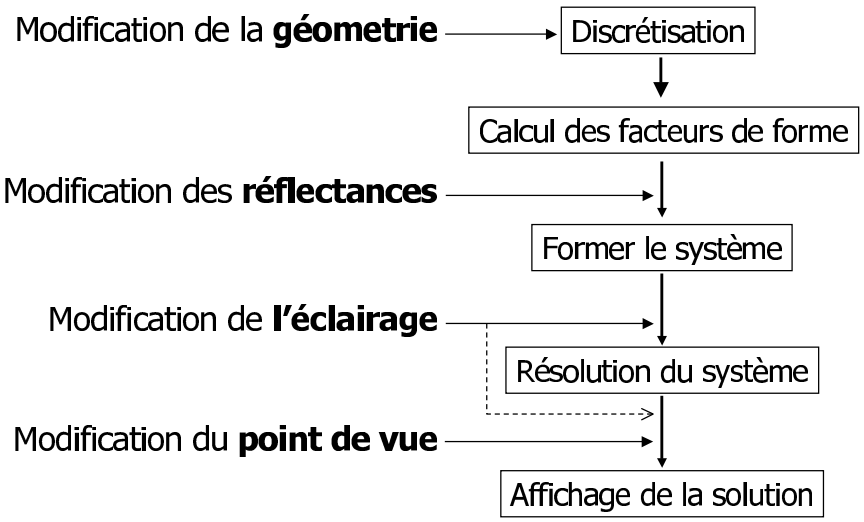


Final gather (récolte finale)

- Lancer de rayons depuis le point de vue
  Re-calcul en chaque point de la visibilité et de l'éclairement

- Ombres parfaites, mais
  Très coûteuses
  Dépendant du point de vue

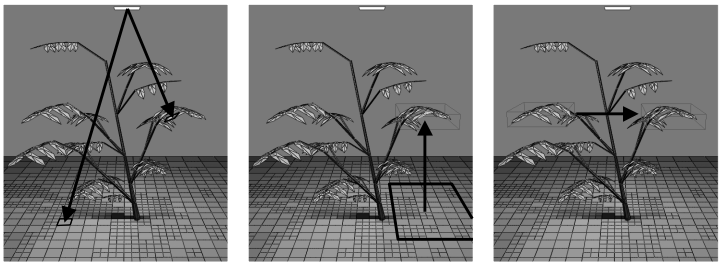
Ordre des opérations



Radiosité - Plan

- Introduction, notations et hypothèses
- Discrétisation du problème
- Résolution de l'équation matricielle
- Méthode hiérarchique
  - Accélerer les temps de calculs
  - Ne propager l'énergie que de là où il y en a
  - Regrouper les échanges d'énergie

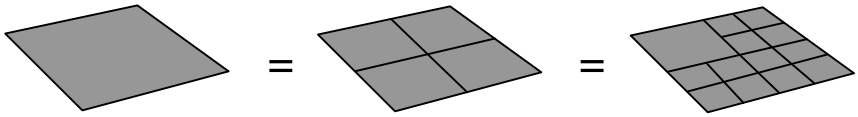
Radiosité hiérarchique



- Etablir les transferts d'énergie (liens)
  - Suffisamment haut pour économiser des calculs
  - Suffisamment bas pour conserver la précision
- Trois phases par itération
  - Raffinement ⇒ assure la complétude des échanges
  - *Gather* ⇒ transfert de l'énergie le long des liens
  - *Push/Pull* ⇒ cohérence multi-échelles

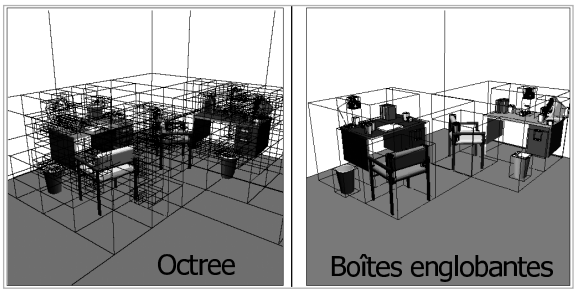
Représentation hiérarchique

- Des surfaces



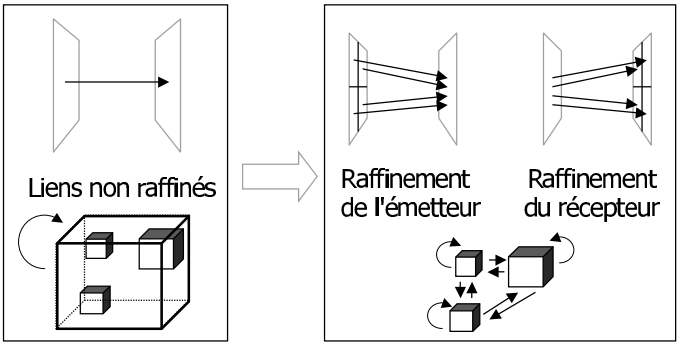
- Des objets

Clusters



Raffinement des liens

- Départ : un unique lien de la scène sur elle même



- Oracle de raffinement
  - Rendre l'erreur uniforme
  - Exemple: BF (radiosité x Facteur de forme)

Push / Pull

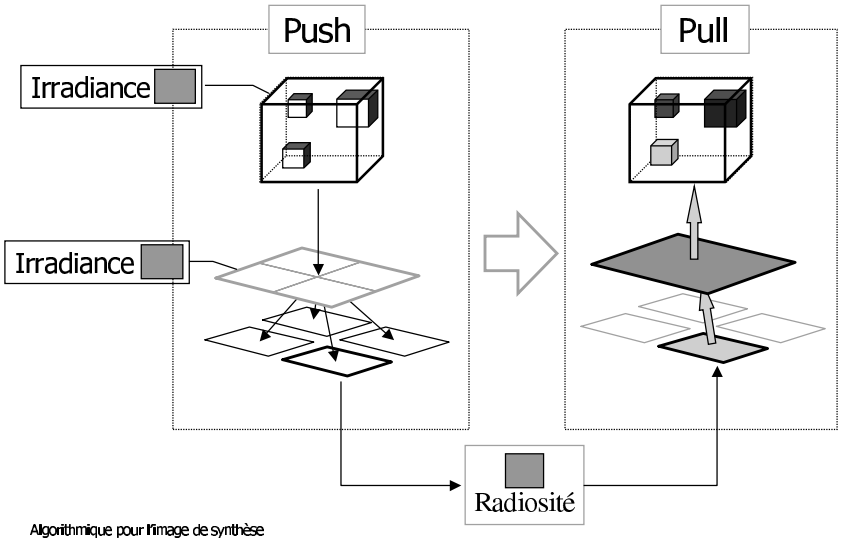
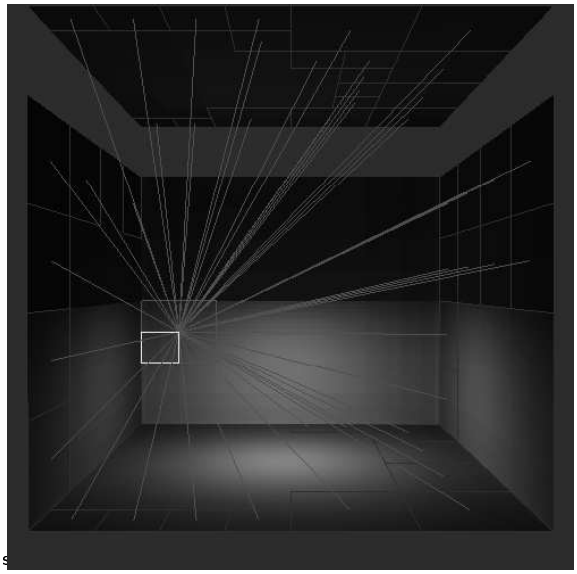
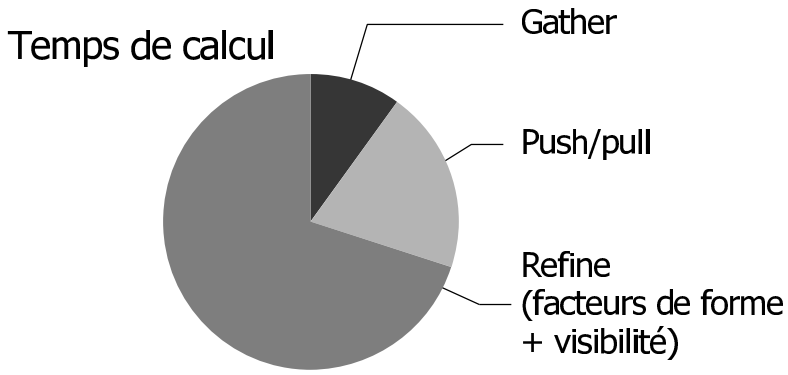


Illustration de la hiérarchie



Répartition des coûts



[Holzschuch94]

Exemples de résultats

Les quaternions

LA représentation des rotations en 3D

Qu'est ce qu'une rotation ?

- Isométrie préservant une droite : l'axe
- Expression matricielle

  Simple pour un axe X, Y ou Z

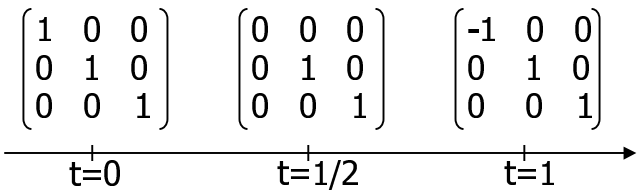
$$R_Z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

  Cas général : une matrice 3x3 orthonormale
- Espace de dimension 3

  9 (matrice 3x3) – 3 (unitaire) – 3 (orthogonale)

Inconvénients

- Passage (axe, angle) $\leftrightarrow$ matrice non trivial
- Imprécisions numériques lors de compositions
  Gram-Schmidt pour re-orthonormaliser
- Comment interpoler entre deux matrices ?
  L'interpolation linéaire détruit l'orthonormalité



Propriétés souhaitées

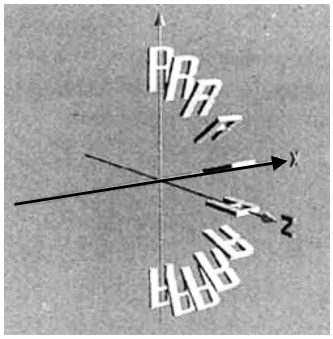
- Interpolation entre orientations
- Stabilité numérique (composition de rotations)
- Expression simple et intuitive
- Efficacité (création, application, modification)
- Convertion matricielle aisée (hardware)

Angles d'Euler

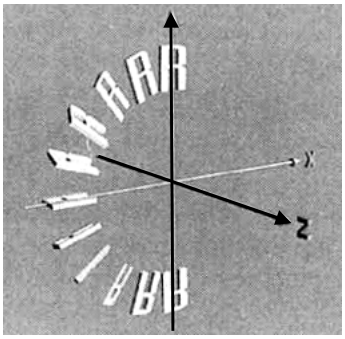
- Extension des coordonnées polaires en 3D
- Rotations successives autour de X, Y puis Z
  - Cet ordre est arbitraire : définir un standard
  - Il suffit que deux axes successifs soient différents
- Espace de dimension 3
- Simple, relativement intuitif

Problème avec les angles d'Euler

- Il n'y a pas unicité de la représentation



Rotation autour de X



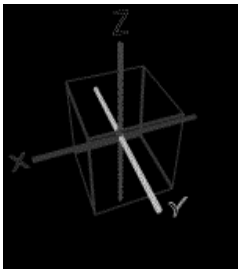
Rotation autour de Y puis Z

- Problème de définition pour l'interpolation

Interpolation des angles d'Euler

- Interpoler les 3 angles
- Trajet dépend de la réprésentation choisie

- Gimbal lock
  - Quand une des rotation passe par 90°
  - On perd un degré de liberté



Représentation par axe et angle

- 3 paramètres (axe normalisé et angle)
- Simple et intuitif
- Interpolation de l'axe et de l'angle
  - Plus de Gimbal lock

- Calculs assez complexes
  - Rotation d'un point (produits vectoriels)
  - Composition de rotations (utiliser les matrices)

- Manque une algèbre propre

Les quaternions

- Extension en 4D des nombres complexes
  Dans $\mathbb{C}$: base (1,i) avec $i^2=-1$
- 4D : base (1,i,j,k) $i^2=j^2=k^2=ijk=-1$ $ij=k$ ↺
- $q = x + yi + zj + wk = (x, y, z, w)$
- Quaternion : vecteur unitaire de $\mathbb{R}^4$
  $x^2 + y^2 + z^2 + w^2 = 1$
- Notation
  $q = v_x + v_y i + v_z j + wk = (v_x, v_y, v_z, w) = (\vec{v}, w)$

Le groupe des quaternions

- On définit le produit interne
  $q_1 = (v_1, w_1)$ $q_2 = (v_2, w_2)$

$$q_1 \bullet q_2 = (w_1 v_2 + w_2 v_1 + v_1 \times v_2 \ , \ w_1 w_2 - v_1 \bullet v_2)$$

Non commutatif

- $q_1 \bullet q_2$ est un quarternion
- $q = (0, 0, 0, 1)$ est l'élément neutre
- Structure de groupe

Quaternion et axe-angle

- On représente une rotation u,θ par

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \sin(\theta/2) \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} \qquad w = \cos(\theta/2)$$

$$|q| = \sin^2(\theta/2)\,|u|^2 + \cos^2(\theta/2) = 1$$

- Avec $q_1 = R_1(u_1, \theta_1)$ et $q_2 = R_2(u_2, \theta_2)$
  $q_1 \bullet q_2$ est la composition de R_1 et R_2 !!

Utilisation des quaternions

- Représentation matricielle

$$R = \begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix}$$

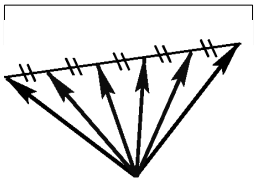
- Rotation inverse : $(\vec{v}, w) \rightarrow (-\vec{v}, w)$
- Rotation d'un point p

$$\mathrm{Rot}_q(p) = q \bullet (p, 0) \bullet q^{-1}$$

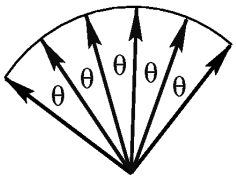
Interpolation de quaternions

- Parfaitement adaptés
- Interpolation sur la sphère unité
- Interpolation linéaire n'est pas unitaire
- Spherical Linear intERPolation

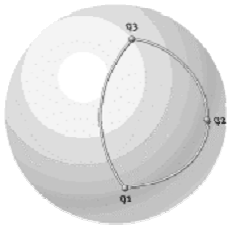
  Vitesse angulaire de rotation constante



Lerping

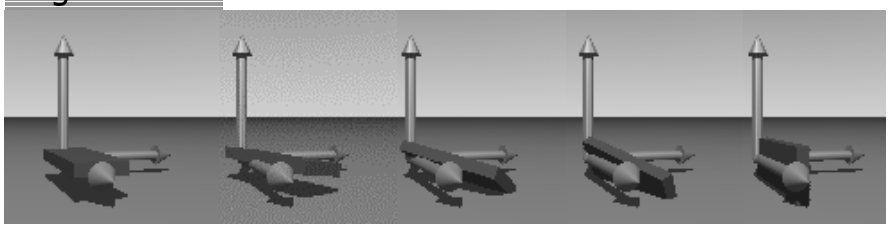


Slerping

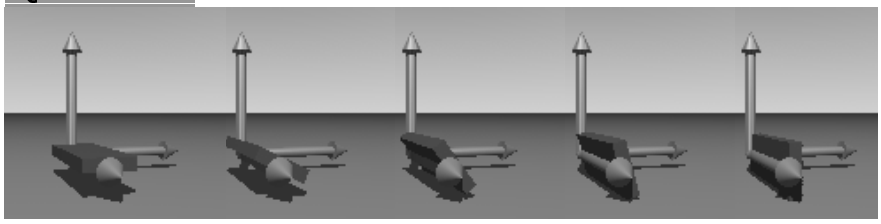


Résultats d'interpolation

Angles d'Euler



Quaternions



Images de Rick Parent

Quaternions - Conclusion

- Formalisme de choix pour les rotations 3D
- Relativement complexes

  Implémentation et mathématiques
- Défini à un signe près (v, w) = (-v, -w)

  Choisir le chemin le plus court sur la sphère
- Extension à nD par l'algèbre de Cliford

$\rightarrow$ Utiliser du code existant

Simplification de maillage

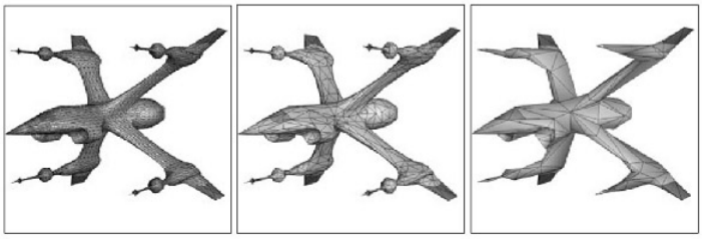
Niveaux de détails

Introduction

- Context of Real-time Rendering
  - A large model to display
  - A fixed amount of time (30 fps)
  - A fixed resolution
- Strategies
  - Display only what is visible
  - Display objects with appropriate details
  - Use simplified version (also for side computations)
    - collision detection
    - shadows
- Optimize rendering

Level of Detail: Principle

- For each object in the scene
  - Store different versions
  - Choose appropriate version
- How to build the simplified versions?
  - How many? Which degree of simplification?
- How to choose appropriate version?



Overview of presentation

Mesh Simplification

Error Metrics

Selection of LOD

An example: QSlim

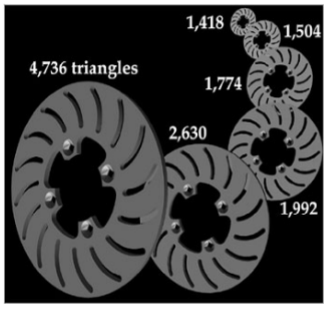
Extensions

Simplification Framework

- Topology preservation
- Local operators
  - work on vertices, edges, faces
  - locally decrease polygon count
- Global operators
  - treat the object as a whole
  - more like "resampling"

Topology preservation

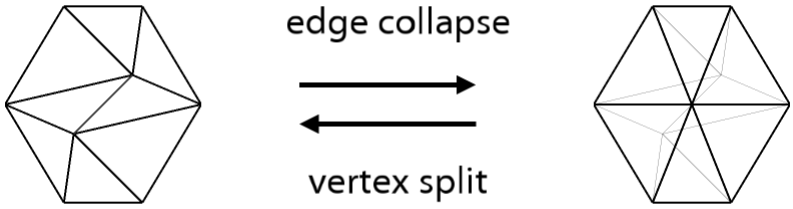
- Maintain Genus
- Can not merge parts
- Limit the simplification
- Not always required



Local operators

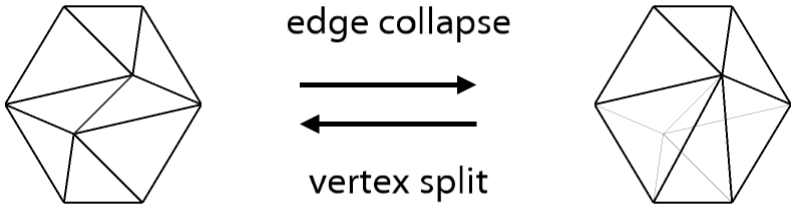
- Edge Collapse
- Vertex pair collapse
- Vertex removal
- Cell collapse

Edge Collapse



Where to place the new vertex?

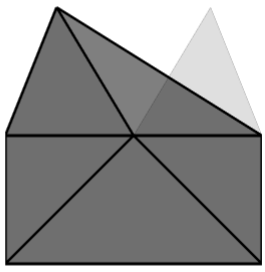
Edge Collapse



- On the edge
  - leaves freedom (but where?)
  - introduces a new point
- On one extremity
  - lower quality
  - no new point
  - ***half-edge collapse***

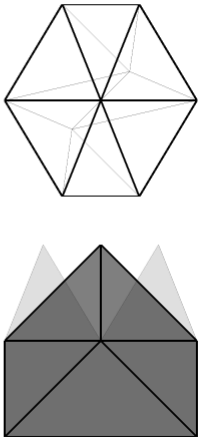
Vertex Pair Collapse

- Generalize edge collapse
  - Collapse any pair of vertices
  - Must choose new position
- Can not consider all pairs
  - avoids quadratic complexity
  - selects "close" vertices
  - achieves linear complexity in practice



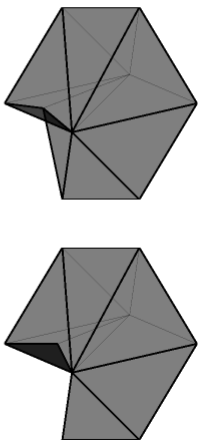
Collapse Considerations

- Post collapse operations
  - Removal of degenerated faces
  - Update of adjacency relations



Collapse Considerations

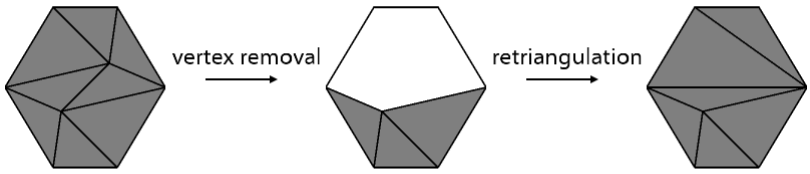
- Post collapse operations
  - Removal of degenerated faces
  - Update of adjacency relations
- Pre collapse operation
  - Avoid mesh fold-over
  - Avoid topology inconstitency



Local operators

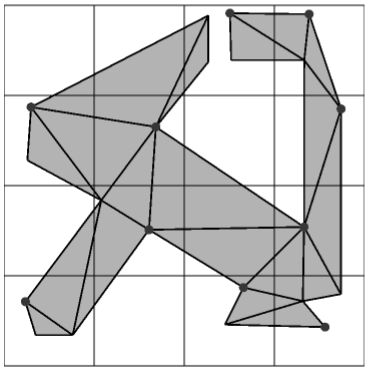
- Edge Collapse
- Vertex pair collapse
- Vertex removal
- Cell collapse

Vertex Removal



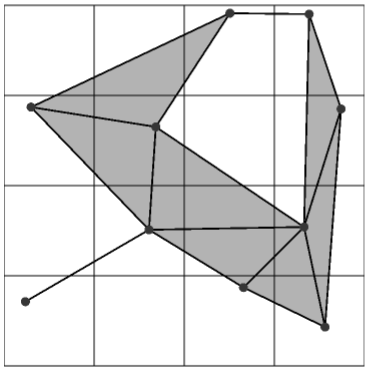
- "Filling the hole" can be tricky
  - Polygon is not "planar"
- Many triangulation
  - Encompass half edge collapse
- Generalized by "polygon merging"

Cell collapse



- Place the model in a grid
- Choose a representant per cell
  - one of the point
  - center of the cell, barycenter,..

Cell collapse



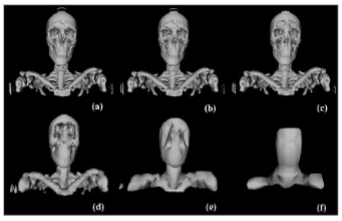
- Place the model in a grid
- Choose a representant per cell
  - one of the point
  - center of the cell, barycenter,..
- Collapse cell's points on representant
- Clean degeneracies
  - remove empty triangles
  - create segments and points

Simplification Framework

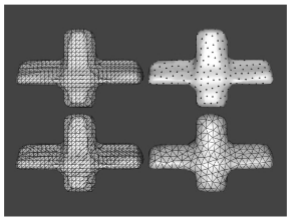
- Topology preservation
- Local operators
  - work on vertices, edges, faces
  - locally decrease polygon count
- Global operators
  - treat the object as a whole
  - more like "resampling"

Global operators

- Volume Rendering [He95]
  - Rasterize the mesh in a grid
  - Low-pass filter the grid
  - Reconstruct the mesh (marching cube)



- Re-tiling polygonal surfaces [Turk92]
  - Distribute point on the surface
  - Re-triangulate the points



Morphological operators

- "Erase" small details
- Merge disconnected parts
- Cannot simplify past some point
- Generalized with alpha hulls [ElSana98]

Overview of presentation

Mesh Simplification

Error Metrics

Selection of LOD

An example: QSlim

Extensions

Error Metrics

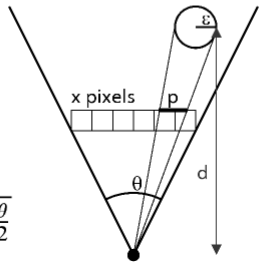
- Why measure Error?
  - Guide the simplification process
  - Know the quality of the results
  - Know when to show a particular LOD
  - Balance quality over a scene
- Key elements
  - Geometric Error
  - Attribute Error
- Incremental vs. Total Error

Error Metrics - geometric error

- How to measure the distance between two surfaces?
  - Hausdorff distance
  - Approximations
- Maximum vs. Average
- Screen Space Error

$$h(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\|$$

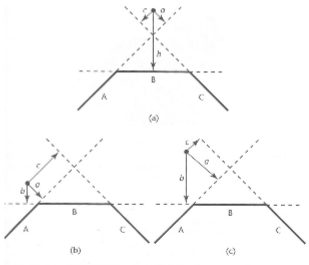
$$H(A,B) = \max(h(A,B), h(B,A))$$



$$p = \frac{\varepsilon\, x}{2d \tan\frac{\theta}{2}}$$

Measuring distance: approaches

- Vertex-Vertex
  - natural for cell collapses
  - miss surface changes
- Vertex-Plane
  - distance to supporting faces
  - innacurate in theory
- Vertex-Surface
  - distance to closest point
  - Progressive Meshes [Hoppe93]
- Surface-Surface
  - strongest error bound
  - hard to compute
  - Simplification Enveloppes [Cohen96]



Overview of presentation

Mesh Simplification

Error Metrics

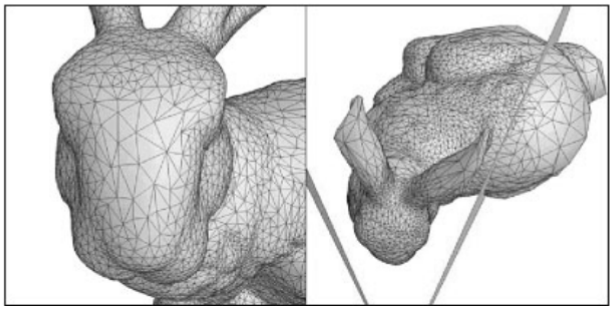
Selection of LOD

An example: QSlim

Extensions

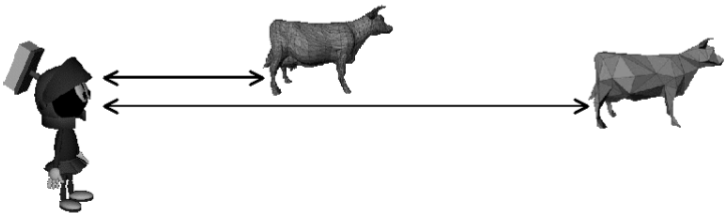
LOD Selection

- LOD type
  - discrete, continuous, view-dependent



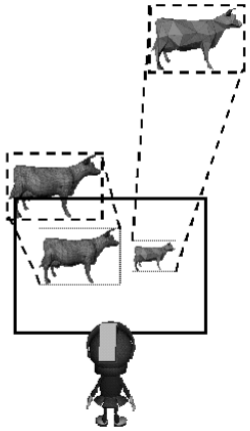
LOD Selection

- LOD type
  - discrete, continuous, view-dependent
- Selection criteria
  - distance
  - screen size



LOD Selection

- LOD type
  - discrete, continuous, view-dependent
- Selection criteria
  - distance
  - screen size



LOD Selection

- LOD type
  - discrete, continuous, view-dependent
- Selection criteria
  - distance
  - screen size
  - priority, hysteresis, environmental conditions, perceptual factors
- Blending Between Transitions
  - alpha blending
  - geomorphs

Overview of presentation

Mesh Simplification

Error Metrics

Selection of LOD

An example: QSlim

Extensions

QSlim algorithm

- For each vertex compute initial quadric
- Build all pairs of "close" vertices
- For each pair:
  - find best position for collapsing
    - if possible, inverse matrix Q+Q'
    - otherwise, test extremities and middle point
  - compute associated error
- Choose collapsing with smallest error
  - compute new quadric for collapsed vertex
  - update list of pairs
- Iterate until:
  - given number of faces reached
  - given error reached

Overview of presentation

Mesh Simplification

Error Metrics

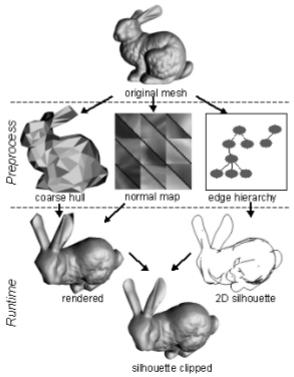
Selection of LOD

An example: QSlim

Extensions

Silhouette Clipping [Sander00]

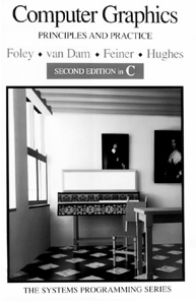
- Silouhette is visually important
- Mesh simplification degrades silouhette



Alternatives representations

- Hard to represent a shape with few polygons
- An image can convey many details!
- Use Image Based Representations (IBR)!
- Imposters [Maciel95,Decoret99]
  - replace distant parts with a textured quad
  - looks nice when static
  - looks flat when moving
- Billboard Clouds [Decoret03]
  - approximate roughly shape with few planes
  - use semi-transparent textures for finer details
  - generalize classic tricks for games (trees)

Bibliographie (1/3)



- Foley, van Dam, Feiner, Hughes
Computer Graphics, Addison-Wesley

- Présentation des API, détails sur OpenGL
- Code à copier-coller pour les différentes API

`artis.imag.fr/~Gilles.Debunne/Enseignement`

- libQGLViewer : viewer 3D facile

`artis.imag.fr/~Gilles.Debunne/QGLViewer`

Bibliographie (2/3)

- Graustein, W. C.

Homogeneous Cartesian Coordinates. Linear Dependence of Points and Lines.

Ch. 3 in Introduction to Higher Geometry, pp. 29-49, 1930.

- E.W. Weisstein, *Books about Quaternions*

`www.ericweisstein.com/encyclopedias/books/Quaternions.html`

- Interpolation de quaternions

Ken Shoemake in SIGGRAPH '85

Bibliographie (3/3)

- F. Sillion

A unified hierarchical algorithm for global illumination with scattering volumes and object clusters.

IEEE Transactions On Graphics. 1(3), sept 1995.

- Niveaux de détail

`http://www.lodbook.com`

- M. Garland and P. Heckbert. *Surface simplication using quadric error metrics*. 1997.

Arithmétique des ordinateurs

Responsable : Nathalie Revol.

Conférenciers : Nathalie Revol, Arnaud Tisserand, Paul Zimmerman

Sommaire

Introduction à l'arithmétique par intervalles

Nathalie Revol

INRIA, projet Arénaire
LIP (UMR CNRS/INRIA/ENSL/UCBL)
46, allée d'Italie, F-69364 Lyon cedex, France
Nathalie.Revol@ens-lyon.fr

Mots clés. Arithmétique par intervalles, calcul garanti, information globale, grossissement des résultats, itération contractante, optimisation globale, algorithme de Hansen, algorithme de Newton par intervalles, résolution de systèmes linéaires par intervalles, résolution de contraintes continues.

1 Historique succinct et subjectif

La naissance de l'arithmétique par intervalles n'est pas datée avec certitude : pour une large partie de la communauté, son père est Ramon Moore qui l'a mentionnée pour la première fois en 1962 dans [30] et qui l'a définie de façon très complète et publiée en 1966 dans [31]. Pour d'autres, on trouve déjà dans un article de T. Sunaga daté de 1958 [53] les fondements de l'arithmétique par intervalles. Dans [17], l'origine de l'arithmétique par intervalles est attribuée à Rosalind Cecil Young qui l'aurait proposée dans sa thèse de doctorat à l'Université de Cambridge en 1931 [56]. Il est très facile de se procurer ces références, grâce au site `http://www.cs.utep.edu/interval-comp/`, rubrique *Early papers*.

Que l'arithmétique par intervalles soit née en 1931, en 1958 ou en 1962, son enfance se prolonge jusqu'à la fin des années 1970 : à ses débuts en effet, cette arithmétique semble rester assez confidentielle. J'en ignore les raisons et ne peux que soulever des hypothèses. La première est que le calcul scientifique ne dispose pas encore d'assez de puissance pour pouvoir se permettre de perdre un facteur de vitesse d'au moins 4 et de mémoire d'au moins 2 en changeant d'arithmétique. La seconde est que l'arithmétique flottante n'est pas encore assez bien spécifiée (arrondis en particulier) pour qu'implanter l'arithmétique par intervalles à partir de l'arithmétique flottante disponible alors soit une tâche aisée.

À partir des années 1980, l'arithmétique par intervalles prend son essor, en Allemagne particulièrement, sous l'impulsion d'U. Kulisch à Karlsruhe. Il réussit à convaincre Nixdorf de développer un processeur spécifique, puis IBM de mettre

au point un jeu d'instructions et un compilateur [19] intégrant l'arithmétique par intervalles. Il va également former un grand nombre d'étudiants qui ont maintenant essaimé dans toute l'Allemagne et contribuent à la maintenir dans le peloton de tête des pays intervallistes.
À la même époque, l'arithmétique flottante voit naître la norme IEEE-754 (voir le cours d'Arnaud Tisserand) qui spécifie en particulier les modes d'arrondis et facilite l'implantation de l'arithmétique par intervalles. Cependant, l'arithmétique par intervalles peine à convaincre des utilisateurs potentiels. Il me semble qu'elle n'a pas répondu à des espoirs, à vrai dire injustifiés, quant à son utilisation comme outil de validation numérique de calculs flottants : en effet, tout calcul par intervalle est un calcul *garanti*, c'est-à-dire que le résultat calculé est un intervalle garanti contenir la valeur ou l'ensemble de valeurs cherché. On s'imaginait alors qu'il suffisait de remplacer le type `float` ou `double` (ou `real...`) par le type `interval` dans un programme pour obtenir un résultat donnant un encadrement fin des erreurs d'arrondi, ce qui n'est pas le cas comme nous le verrons. Cet échec a desservi l'arithmétique par intervalles à ses débuts.

L'arithmétique par intervalles continue à se développer, mais avec des objectifs différents, depuis le début des années 1990. Son atout majeur, qui est de permettre de calculer sur des ensembles, est désormais mis à profit : c'est par exemple le seul outil déterministe (à ma connaissance) permettant de déterminer l'optimum global d'une fonction continue [36], de déterminer tous les zéros d'une fonction et de prouver en même temps leur existence et leur éventuelle unicité ou encore de déterminer l'image directe ou inverse d'un ensemble par une fonction [20].

2 Calcul par intervalles

2.1 Intervalles

En arithmétique par intervalles, on ne manipule plus des nombres, qui approchent plus ou moins fidèlement une valeur, mais des intervalles contenant cette valeur. Par exemple, on peut tenir compte d'une erreur de mesure en remplaçant une valeur mesurée x avec une incertitude ε par l'intervalle $[x - \varepsilon, x + \varepsilon]$. On peut également remplacer une valeur non exactement représentable, telle que π, par un intervalle la contenant ; si l'on dispose d'un ordinateur représentant les nombres en base 10 avec 3 chiffres, π sera remplacé par $[3.14, 3.15]$. Enfin, si l'on désire obtenir un résultat valide pour tout un ensemble de valeurs, on utilise un intervalle contenant ces valeurs. En effet, l'objectif de l'arithmétique par intervalles est de fournir des résultats qui contiennent à coup sûr la valeur ou l'ensemble cherché ; on parle alors de résultats *garantis* ou *validés*, ou encore *certifiés*.

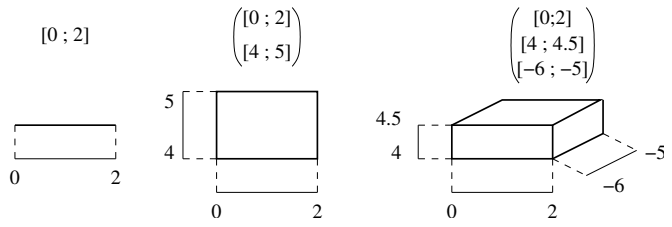


FIG. 6 – Des exemples de vecteur intervalle en dimensions 1, 2 et 3.

Comme cela a été implicitement admis jusqu'à présent, les intervalles sont des sous-ensembles fermés connexes de $\mathbb{R}$. On notera $\mathbb{IR}$ l'ensemble des intervalles de $\mathbb{R}$. On peut les généraliser en plusieurs dimensions : un vecteur intervalle $\boldsymbol{x} \in \mathbb{IR}^n$ est un vecteur dont les n composantes sont des intervalles et une matrice intervalle $\boldsymbol{A} \in \mathbb{IR}^{m\times n}$ est une matrice dont les composantes sont des intervalles. Une représentation graphique d'un vecteur de $\mathbb{IR}$, $\mathbb{IR}^2$ et $\mathbb{IR}^3$ est donnée figure 6. Elle illustre le fait qu'un vecteur intervalle est un ensemble parallélépipédique de vecteurs aux côtés parallèles aux axes du repère ; cela justifie que par la suite on utilisera indifféremment les termes de vecteur intervalle, de pavé ou de boîte ou même d'intervalle.

Notations. Les objets intervalles seront désignés par des caractères gras : $\boldsymbol{x}$. On notera $\underline{x}$ le minimum de $\boldsymbol{x}$ et $\overline{x}$ son maximum, avec l'ordre partiel sur $\mathbb{R}^n$: $x \leq y$ ssi $x_i \leq y_i$ pour $1 \leq i \leq n$. On a alors $\boldsymbol{x} = [\underline{x}, \overline{x}]$. Enfin, $w(\boldsymbol{x})$ est la largeur de $\boldsymbol{x}$: $\overline{x} - \underline{x}$ (avec w pour *width*) ou encore son diamètre. Le centre $mid(\boldsymbol{x})$ et son rayon $rad(\boldsymbol{x})$ sont définis par $mid(\boldsymbol{x}) = (\underline{x} + \overline{x})/2$ et $rad(\boldsymbol{x}) = (\overline{x} - \underline{x})/2$ $= 1/2w(\boldsymbol{x})$. On désignera par les adjectifs " scalaire" ou " ponctuel" un objet numérique usuel et on le confondra avec l'intervalle de largeur 0 ne contenant que cette valeur.

2.2 Calculs

2.2.1 Définition (abstraite)

Le résultat d'une opération entre deux intervalles : $\boldsymbol{x} \diamond \boldsymbol{y}$, resp. d'une fonction $f(\boldsymbol{z})$, est le plus petit intervalle (ou vecteur intervalle) contenant

$$\{x \diamond y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\}, \text{ resp. } \{f(z) \mid z \in \boldsymbol{z}\},$$

c'est-à-dire le plus petit intervalle contenant tous les résultats possibles de l'opération appliquée à tous les éléments x de $\boldsymbol{x}$ et tous les éléments y de $\boldsymbol{y}$, resp. tous les résultats possibles de f appliquée à tous les éléments z de $\boldsymbol{z}$.

2.2.2 Opérations arithmétiques

Quand on applique la définition précédente aux opérations arithmétiques $=$, $-$, $\times$, ^{2}, $/$ ou $\sqrt{\ }$, on obtient les formules suivantes, plus utilisables en pratique que la définition abstraite :

$$\begin{array}{lcl}
[\underline{x},\,\overline{x}] + [\underline{y},\,\overline{y}] & = & [\underline{x}+\underline{y},\,\overline{x}+\overline{y}] \\
[\underline{x},\,\overline{x}] - [\underline{y},\,\overline{y}] & = & [\underline{x}-\overline{y},\,\overline{x}-\underline{y}] \\
[\underline{x},\,\overline{x}] \times [\underline{y},\,\overline{y}] & = & [\min(\underline{x}*\underline{y},\underline{x}*\overline{y},\,\overline{x}*\underline{y},\,\overline{x}*\overline{y}), \max(\underline{x}*\underline{y},\underline{x}*\overline{y},\,\overline{x}*\underline{y},\,\overline{x}*\overline{y})] \\
[\underline{x},\,\overline{x}]^2 & = & [\min(\underline{x}^2,\,\overline{x}^2), \max(\underline{x}^2,\,\overline{x}^2)] \text{ si } 0 \notin [\underline{x},\,\overline{x}] \\
 & & \text{et } [0, \max(\underline{x}^2,\,\overline{x}^2)] \text{ sinon} \\
1/\,[\underline{y},\,\overline{y}] & = & [\min(1/\underline{y}, 1/\overline{y}), \max(1/\underline{y}, 1/\overline{y})] \text{ si } 0 \notin [\underline{y},\,\overline{y}] \\
[\underline{x},\,\overline{x}]\,/\,[\underline{y},\,\overline{y}] & = & [\underline{x},\,\overline{x}] \times (1/\,[\underline{y},\,\overline{y}]) \text{ si } 0 \notin [\underline{y},\,\overline{y}] \\
\sqrt{[\underline{x},\,\overline{x}]} & = & [\sqrt{\underline{x}},\sqrt{\overline{x}}] \text{ si } 0 \leq \underline{x}
\end{array}$$

On obtient ces formules en utilisant la monotonie (au moins partielle) de ces opérations.

2.2.3 Propriétés algébriques

On peut d'ores et déjà constater que les opérations définies ci-dessus ne présentent pas les propriétés algébriques de leurs contreparties ponctuelles. Tout d'abord, la soustraction n'est pas la réciproque de l'addition. Par exemple, si $\boldsymbol{x} = [2,3]$, $\boldsymbol{x} - \boldsymbol{x} = [2,3] - [2,3] = [-1,1] \neq 0$ même s'il le contient. En effet,

$$\boldsymbol{x} - \boldsymbol{x} = \{x - y \mid x \in \boldsymbol{x},\, y \in \boldsymbol{x}\} \supset \{x - x \mid x \in \boldsymbol{x}\} = \{0\}$$

et l'inclusion est stricte.
De la même façon, la division n'est pas la réciproque de la multiplication : si $\boldsymbol{x} = [2,3]$, l'intervalle $\boldsymbol{x}/\boldsymbol{x} = [2,3]/[2,3] = [2/3, 3/2]$ n'est pas égal à 1 même s'il le contient.
De plus, la multiplication d'un intervalle par lui-même n'est pas égal à l'élévation au carré : si $\boldsymbol{x} = [-3,2]$,

$$\boldsymbol{x} \times \boldsymbol{x} = [-3,2] \times [-3,2] = [-6,9]$$

alors que

$$\boldsymbol{x}^2 = \{x^2 \mid x \in \boldsymbol{x}\} = [0,9]\,.$$

Enfin, la multiplication n'est pas distributive par rapport à l'addition : si $\boldsymbol{x} = [-2, 3]$, $\boldsymbol{y} = [1, 4]$ et $\boldsymbol{z} = [-2, 1]$,

$$\begin{array}{rcl} \boldsymbol{x} \times (\boldsymbol{y} + \boldsymbol{z}) & = & [-2, 3] \times ([1, 4] + [-2, 1]) \\ & = & [-2, 3] \times [-1, 5] \\ & = & [-10, 15] \\ \boldsymbol{x} \times \boldsymbol{y} + \boldsymbol{x} \times \boldsymbol{z} & = & [-2, 3] \times [1, 4] + [-2, 3] \times [-2, 1] \\ & = & [-8, 12] + [-6, 4] \\ & = & [-14, 16] \end{array}$$

Comme l'illustre cet exemple, la multiplication est sous-distributive par rapport à l'addition, c'est-à-dire que

$$\boldsymbol{x} \times (\boldsymbol{y} + \boldsymbol{z}) \subset \boldsymbol{x} \times \boldsymbol{y} + \boldsymbol{x} \times \boldsymbol{z}.$$

La raison en est toujours que dans le premier cas on détermine $\{x \times (y + z) \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}, z \in \boldsymbol{z}\}$ alors que dans le second on calcule $\{x \times y + x' \times z \mid x \in \boldsymbol{x}, x' \in \boldsymbol{x}, y \in \boldsymbol{y}, z \in \boldsymbol{z}\}$, autrement dit on n'a pas identité de x et x'.

2.2.4 Fonctions élémentaires

On peut également définir des fonctions élémentaires ($\sin$, $\exp$, $\operatorname{acoth}\ldots$) prenant des intervalles pour argument, à l'aide de la définition abstraite ci-dessus. Pour les fonctions monotones telles que l'exponentielle ou l'arc-cotangente hyperbolique, on déduit facilement les formules permettant de les calculer :

$$\begin{array}{rcl} \exp([\underline{x}, \overline{x}]) & = & [\exp \underline{x}, \exp \overline{x}] \\ & & \text{puisque } \exp \text{ est croissante,} \\ \operatorname{acoth}([\underline{x}, \overline{x}]) & = & [\operatorname{acoth} \overline{x}, \operatorname{acoth} \underline{x}] \text{ si } [\underline{x}, \overline{x}] \not\ni 0, \\ & & \text{puisque } \operatorname{acoth} \text{ est décroissante sur } \mathbb{R}^{+*} \text{ et } \mathbb{R}^{-*}. \end{array}$$

En revanche, il faut être plus soigneux pour les fonctions périodiques par exemple, mais il est possible d'établir des algorithmes de calcul pour ces fonctions, dès que l'on dispose de ces fonctions sur les réels. Par exemple, $\sin[\pi/3, \pi] = [0, 1]$.
Enfin, on ne sait définir les fonctions élémentaires que sur des intervalles inclus dans leur domaine de définition : on a vu ci-dessus que acoth n'était définie que pour des intervalles ne contenant pas 0, de la même manière le logarithme ne sera défini que pour des intervalles strictement positifs[8].

[8] C'est l'une des options possibles, l'autre étant de définir $f(\boldsymbol{x})$ comme $\{f(x) \mid x \in \boldsymbol{x} \cap \mathcal{D}_f\}$ où $\mathcal{D}_f$ est le domaine de définition, comme dans [20] ou [52].

2.3 Évaluation d'une expression

Puisque l'on sait calculer le résultat d'une opération arithmétique ou d'une fonction élémentaire quand les variables prennent pour valeur des intervalles, on sait également calculer le résultat d'une expression mêlant opérations arithmétiques ou algébriques et fonctions élémentaires sur des intervalles. Voici simplement quelques exemples pour illustrer ce propos.
L'expression polynomiale $\boldsymbol{x}^3 - 2\boldsymbol{x}^2 + \boldsymbol{x} - 3$, avec $\boldsymbol{x} = [-5, 2]$, a pour résultat

$$\begin{array}{ll} & [-5,2]^3 - 2\,[-5,2]^2 + [-5,2] - 3 \\ = & [-125,8] - 2\,[0,25] + [-5,2] - 3 \\ = & [-125,8] - [0,50] + [-5,2] - 3 \\ = & [-183,7]\,. \end{array}$$

L'expression en plusieurs variables $\sin \boldsymbol{x} + 2\boldsymbol{x} \exp \boldsymbol{y} - \boldsymbol{y}^2\sqrt{\boldsymbol{z}}$ avec $\boldsymbol{x} = [-\pi, \pi/4]$, $\boldsymbol{y} = [-1, 1]$ et $\boldsymbol{z} = [1, 4]$ a pour résultat

$$\begin{array}{ll} & \sin[-\pi,\pi/4] + 2\,[-\pi,\pi/4] \times \exp[-1,1] - [-1,1]^2 \times \sqrt{[1,4]} \\ = & \left[-1, \sqrt{2}/2\right] + [-2\pi, \pi/2] \times [1/e, e] - [0,1] \times [1,2] \\ = & \left[-1, \sqrt{2}/2\right] + [-2\pi e, \pi e/2] - [0,2] \\ = & \left[-3 - 2\pi e, \sqrt{2}/2 + \pi e/2\right] \end{array}$$

2.4 Implantations sur ordinateur

2.4.1 Comment implanter

Pour pouvoir implanter l'arithmétique par intervalles sur ordinateur, il faut être capable de retourner toujours un intervalle qui soit à la fois représentable en machine et contenant l'intervalle mathématique défini plus haut. Pour cela, il faut être capable de retourner un intervalle dont le minimum est inférieur ou égal au minimum de l'intervalle exact et dont le maximum est supérieur ou égal au maximum de l'intervalle exact[9]. C'est possible si on dispose d'arrondis dirigés pour le calcul ponctuel ; dans le cas où les intervalles sont représentés par leurs extrémités, on va arrondir vers le bas le résultat du calcul de la borne inférieure, et vers le haut le résultat du calcul de la borne supérieure. En particulier c'est le cas pour les opérations arithmétiques et algébriques définies par la norme IEEE-754 pour l'arithmétique flottante, qui est présentée dans le cours d'Arnaud Tisserand dans cette même école, mais hélas pas encore pour les fonctions élémentaires. On peut

[9]Ceci doit être vrai quelle que soit la représentation interne des intervalles, que ce soit par leurs extrémités, ou par leur centre et leur rayon ou toute autre représentation que vous pourrez imaginer.

donc implanter les opérations arithmétiques en utilisant les modes d'arrondi disponibles (même s'ils sont plus ou moins facilement accessibles) et se retrousser les manches pour les fonctions élémentaires, ou utiliser une bibliothèque qui les calcule avec arrondis dirigés, comme Crlibm (cf. `http://perso.ens-lyon.fr/catherine.daramy/crlibm.html`) développée dans le projet Arénaire, ou MPFR (cf. `http://www.mpfr.org/`) du projet Spaces, Paul Zimmermann étant à l'origine de MPFR et un développeur actif. Dans le cas où un intervalle est représenté par son centre et son rayon, il est possible, toujours grâce à l'arithmétique IEEE-754, de déterminer le centre du résultat, généralement en utilisant l'arrondi au plus près, puis le rayon du résultat en utilisant le mode d'arrondi vers le haut ainsi que des informations sur l'erreur commise en calculant le centre.

2.4.2 Quelques implantations

Mentionnons tout d'abord, pour en rester à la représentation par centre et rayon, le paquetage IntLab [49] utilisable en MatLab. Ce paquetage est développé et distribué par S. Rump. Cette représentation permet de calculer le centre de la matrice résultat (puisqu'en MatLab, tout est matrice) sans changer de mode d'arrondi pendant le calcul (on reste en arrondi au plus près) et les calculs matriciels optimisés de MatLab sont utilisés, puis de passer en arrondi vers le haut pour calculer le rayon, à nouveau en profitant des calculs matriciels rapides. En effet, tout changement de mode d'arrondi en cours de calcul ralentit le calcul (il faut vider le pipeline...). L'inconvénient de cette représentation est de conduire à des intervalles plus larges que ceux obtenus avec la représentation d'un intervalle par ses extrémités.

La grande majorité des implantations de l'arithmétique par intervalles utilisent en effet la représentation par extrémités. Elles utilisent très souvent aussi l'arithmétique flottante simple ou double précision disponible sur les processeurs. Séparons ces implantations en trois grandes catégories : compilateurs, bibliothèques et logiciels de plus haut niveau.

Du côté des compilateurs, IBM s'est laissé convaincre le premier, dans les années 1980, de développer une extension du langage Fortran 77, nommée ACRITH [19] et du compilateur qui allait avec. Cependant, les débuts de l'arithmétique par intervalles ayant été difficiles, IBM a renoncé. Actuellement, Sun propose dans ses compilateurs Sun Forte pour Fortran 95 [52] et C++ [51] une extension à l'arithmétique par intervalles : type intervalle, opérations arithmétiques, fonctions élémentaires, opérations ensemblistes telles que l'union ou l'intersection.

Les bibliothèques sont plus nombreuses. Les bibliothèques XSC, pour *language eXtension for Scientific Computing*, à savoir Pascal-XSC, C-XSC et C++-XC [23], ont pris la relève de ACRITH pour ces langages : type intervalle, opérations arithmétiques, fonctions élémentaires, vecteurs et matrices. La bibliothèque BIAS/-PROFIL [25] comporte une couche basse en C, implantant les types simple, vecteur et matrice intervalles ainsi que les opérations arithmétiques sur ces types ; cette couche basse s'appelle BIAS pour *Basic Interval Algebra Subroutines*, par analogie avec les BLAS : *Basic Linear Algebra Subroutines*. Les fonctions élémentaires de BIAS ne sont pas garanties correctes. La couche haute, PROFIL [24], est écrite en C++ et contient des mécanismes de différentiation automatique : calcul des gradients et des Hessiennes en mode *forward*. Malheureusement cette bibliothèque est un peu ancienne et le C++ utilisé ne compile plus.
La bibliothèque fi_lib [27] est une bibliothèque en C dont les deux particularités sont de ne jamais changer le mode d'arrondi (et de fonctionner correctement pour tous les modes d'arrondi IEEE-754, sans savoir lequel est en usage) et de calculer correctement les fonctions élémentaires : celles-ci ont été complètement écrites et tiennent compte des erreurs d'approximation de ces fonctions et des erreurs d'arrondi. Enfin, un des grands noms de l'arithmétique par intervalles, B. Kearfott, propose également la bibliothèque intlib, écrite en Fortran 90. La bibliothèque intlib met en place un mécanisme de différentiation automatique en mode *backward* et la gestion des branchements conditionnels. Grâce à intlib, B. Kearfott a développé et implanté de nombreux algorithmes et les a rendus efficaces en pratique, par ce constant retour de la pratique sur les développements théoriques [1].

On peut également trouver l'arithmétique par intervalles dans des logiciels non dédiés à cette arithmétique, comme Aquarels(Atelier de QUAlité numérique pour la REalisation de Logiciels Scientifiques) [35] développé il y a quelques années à l'IRISA, Prolog IV [7] qui utilise les intervalles pour la résolution de contraintes numériques, Numerica [55] qui était entièrement dédié à la résolution garantie de contraintes numériques (sa commercialisation a cessé), ou Alias [12], un solveur basé sur l'arithmétique par intervalles, dont le développement est encore en cours et qui ne cesse de s'enrichir de nouvelles méthodes.

Pour terminer, citons des paquetages ou bibliothèques d'arithmétique par intervalles en précision arbitraire, c'est-à-dire qui représentent les intervalles à l'aide de nombres rationnels exacts ou flottants en précision quelconque. Des paquetages utilisant les flottants en précision arbitraire existent en Maple : intpakX [15] et en Mathematica [22]. Cependant, les résultats calculés ont beau avoir beaucoup de chiffres, ils ne sont pas garantis puisque l'arithmétique flottante de ces deux systèmes est mal spécifiée. Arithmos [9] propose une représentation à l'aide de rationnels (exacts). Enfin MPFI [41] est une bibliothèque en C/C++ qui utilise les

flottants en précision arbitraire de MPFR [26] pour représenter les intervalles et calculer des résultats garantis et précis.

3 Forces et faiblesses de l'arithmétique par intervalles

3.1 Force : calcul sur des ensembles

L'avantage majeur de l'arithmétique par intervalles est qu'elle permet de calculer avec des ensembles. C'est cette force qui a permis de surmonter la mauvaise impression laissée par des débuts peu concluants (estimation grossière des erreurs d'arrondi), de repartir vers des utilisations adaptées et de faire l'effort de développer des algorithmes qui tiennent compte des avantages et difficultés de l'arithmétique par intervalles.

Si l'on cherche à prouver qu'une valeur donnée n'est jamais atteinte – dans le cas où cette valeur correspond à un comportement dangereux ou à une situation incontrôlable par exemple, voir figure 7 – et que l'évaluation de l'expression sur un intervalle de valeurs d'entrées renvoie un résultat ne contenant pas cette valeur, on a la preuve cherchée. On peut par exemple chercher à déterminer si une sous-expresssion apparaissant au dénominateur d'une expression donnée peut s'annuler. Si l'évaluation de cette sous-expression sur (un intervalle contenant) l'ensemble des valeurs d'entrée considérées renvoie un intervalle ne contenant pas 0, on peut lancer le calcul, ponctuel ou par intervalles, sans craindre de division par 0. On peut également chercher si une zone dangereuse peut être atteinte. Là encore, si le résultat du calcul a une intersection vide avec ladite zone, cette zone est prouvée inaccessible.

L'arithmétique par intervalles, en permettant le calcul sur des ensembles, rend effectifs des théorèmes tels que le théorème de Brouwer/Schauder, sous-jacent dans les résultats du §5.3. Cela signifie qu'en calculant par intervalles, on peut obtenir des preuves, au sens mathématique du terme, de certaines propriétés. On a déjà vu des preuves d'absence de solution, on peut également obtenir des preuves d'existence et d'existence et unicité de solutions, cf. §5.3.

3.2 Force : informations globales, optimisation globale

Savoir calculer sur des ensembles, même si on est restreint à des ensembles de forme particulière, permet également d'obtenir des informations globales précieuses pour l'optimisation globale d'une fonction. Supposons que l'on cherche à

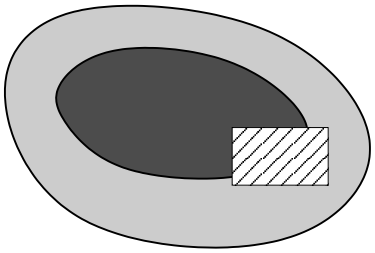


FIG. 7 – La zone gris sombre est la zone de comportement " sûr " et la zone gris clair est la zone de comportement moins sûr mais encore contrôlable. En dehors, le système n'est plus sûr ni contrôlable. La zone calculée (rectangle hachuré) ayant une intersection vide avec la zone non contrôlable, le système n'est pas dangereux.

déterminer le minimum global x^* d'une fonction f de $\mathbb{R}^n$ dans $\mathbb{R}$ assez régulière[10] (au moins continue et si possible de classe $\mathcal{C}^2$) sur un pavé $\boldsymbol{x}_0$, que l'on peut supposer être fourni par l'utilisateur. On suppose aussi que l'on dispose d'une expression de f, afin de pouvoir l'évaluer sur des intervalles. On supposera de plus qu'il s'agit d'un point stationnaire, c'est-à-dire d'un minimum sur lequel le gradient de f s'annule. Autrement dit, on cherche un minimum à l'intérieur de $\boldsymbol{x}_0$, ou encore on ne cherche pas à minimiser une fonction telle que $f(x) = x$ sur un intervalle quelconque.

Voici un premier algorithme, élémentaire, pour déterminer un petit intervalle contenant x^* :

Données : f la fonction à minimiser et $\boldsymbol{x}_0$ l'intervalle dans lequel on cherche le minimum

ε seuil de largeur des intervalles résultats

Résultats : $[\underline{f};\ \overline{f}] \ni f^* = \min_{x \in \boldsymbol{x}_0} f(x)$ // un encadrement de la valeur minimale de f sur $\boldsymbol{x}_0$

Res une liste de pavés $\boldsymbol{x}^*$ tels que $\boldsymbol{f}(\boldsymbol{x}^*) \cap [\underline{f};\ \overline{f}] \neq \emptyset$

Initialisations :

$\mathcal{L} := \{\boldsymbol{x}_0\}$ la liste des pavés en attente de traitement

Res $:= \emptyset$

$[\underline{f};\ \overline{f}] := f(\boldsymbol{x}_0)$

évaluer f en un ou plusieurs points de $\boldsymbol{x}_0$, par exemple $\overline{f} := f(mid(\boldsymbol{x}_0))$

// on a maintenant un encadrement plus précis de $f(x^*)$ par $[\underline{f},\ \overline{f}]$

tant que $\mathcal{L} \neq \emptyset$ **faire**

[10] Dans ce qui suit, on supposera f de classe $\mathcal{C}^2$; si ce n'est pas le cas, il suffit de supprimer, dans les différents algorithmes, les procédures qui nécessitent une telle régularité.

sortir un pavé $\boldsymbol{x}$ de $\mathcal{L}$
couper $\boldsymbol{x}$ en deux : $\boldsymbol{x}_1$ et $\boldsymbol{x}_2$
évaluer $f(\boldsymbol{x}_1) : [\underline{f_1}, \overline{f_1}]$
mettre à jour $\overline{f}$: si $\overline{f_1} \leq \overline{f}$ alors $\overline{f} := \overline{f_1}$
si $\underline{f_1} > \overline{f}$ alors
    jeter $\boldsymbol{x}_1$ // $\boldsymbol{x}_1$ ne peut pas contenir le minimum x^*
sinon si $w(\boldsymbol{x}_1) \leq \varepsilon$ alors ranger $\boldsymbol{x}_1$ dans Res
sinon ranger $\boldsymbol{x}_1$ dans $\mathcal{L}$
idem avec $\boldsymbol{x_2}$

Fin :

mettre à jour l'encadrement de $f(x^*) = [\underline{f}, \overline{f}] : \underline{f} = \min_{y=f(\boldsymbol{x}),x\in Res} \underline{y}$, $\overline{f} = \max_{y=f(\boldsymbol{x}),x\in Res} \overline{y}$
éliminer les $\boldsymbol{x}$ de Res tels que $\underline{y} > \overline{f}$ avec $[\underline{y}, \overline{y}] = f(\boldsymbol{x})$

Cet algorithme renvoie une liste d'intervalles, Res, qui contient tous les intervalles susceptibles de contenir un minimum global : sur chacun de ces intervalles, f prend des valeurs proches de l'optimum. On peut noter une caractéristique de l'arithmétique par intervalles : cet algorithme garantit de n'oublier aucun minimum global. Cet algorithme est aussi le seul envisageable si f est de classe $\mathcal{C}^0$ uniquement.

Si f est plus régulière, on peut appliquer différentes stratégies pour tenter d'éliminer rapidement ou de réduire un pavé. Par réduire $x = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, on entend diminuer au moins l'une des largeurs $w(\boldsymbol{x}_i)$ pour i entre 1 et n.
Les critères de rejet d'un pavé $\boldsymbol{x}$ sont les suivants :

- si $f(\boldsymbol{x}) = [\underline{y}, \overline{y}]$ est trop élevé : $\underline{y} > \overline{f}$ (c'est le cas de $X1$ sur la figure 8) ;
- si grad $f(\boldsymbol{x}) \not\ni 0$: $\boldsymbol{x}$ ne contient pas de point stationnaire (c'est le cas de $X2$ sur la figure 8) ;
- si Hess $f(\boldsymbol{x})$, la Hessienne de f sur $\boldsymbol{x}$, ne contient aucune matrice symétrique positive définie (c'est le cas de $X3$ sur la figure 8) : dans ce cas f n'est pas localement convexe sur $\boldsymbol{x}$; ceci est difficile à tester, en pratique on teste uniquement si Hess $f(\boldsymbol{x})$ contient une matrice à diagonale positive, ce qui est moins souvent vérifié mais plus facile à tester.

Dans tous les cas mentionnés ci-dessus, le pavé $\boldsymbol{x}$ ne peut pas contenir de minimum global et on cesse donc de l'examiner pour passer au prochain pavé à traiter. Si un intervalle $\boldsymbol{x}$ est encore en lice après ces trois tests on tente de le réduire avant de le remettre en attente de traitement et de le couper en deux.

Une première possibilité est de limiter la recherche au sous-pavé $\boldsymbol{x}'$ de $\boldsymbol{x}$ sur lequel $f(\boldsymbol{x}') \leq \overline{f}$: pour cela on peut soit utiliser un développement de Taylor (cf.

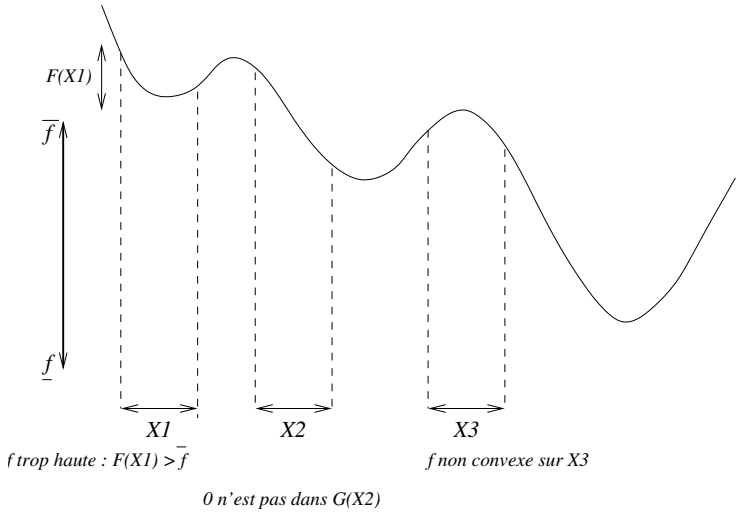


FIG. 8 – Critères permettant de rejeter un intervalle qui ne contient pas le minimum.

§4) d'ordre 1 ou 2 de f et résoudre une inégalité affine ou quadratique, soit appliquer les techniques de résolution de contraintes (propagation et rétro-propagation) présentées au §7. En pratique, ces deux techniques sont rarement mises en œuvre.

Une seconde possibilité consiste à rechercher les zéros du gradient de f sur $\boldsymbol{x}$: on applique une itération ou un petit nombre d'itérations de l'algorithme de Newton par intervalles qui sera expliqué au §5.

L'algorithme d'optimisation globale, dû à Hansen [16], s'écrit donc.

Données :
f et $\boldsymbol{f}$ une extension intervalle de f (cf. §4),
$\boldsymbol{G}$ une extension intervalle de grad$f$,
$\boldsymbol{H}$ une extension intervalle de la Hessienne de f
$\boldsymbol{x}_0$ le pavé de recherche
Résultats : $[\underline{f};\, \overline{f}] \ni f^* = \min_{x \in \boldsymbol{x}_0} f(x)$
Res une liste de pavés $\boldsymbol{x}^*$ tels que $\boldsymbol{f}(\boldsymbol{x}^*) \cap [\underline{f};\, \overline{f}] \neq \emptyset$
Initialisations :
$\mathcal{L} := \{\boldsymbol{x}_0\}$ la liste des pavés en attente de traitement
Res $:= \emptyset$
$[\underline{f};\, \overline{f}] := \boldsymbol{f}(\boldsymbol{x}_0)$
tant que $\mathcal{L} \neq \emptyset$ **faire**
sortir un pavé $\boldsymbol{x}$ de $\mathcal{L}$

$\boldsymbol{x}'$:= réduire $\boldsymbol{x}$:
résoudre $f(\boldsymbol{x}') \leq \overline{f}$ grâce à un développement de Taylor d'ordre 1
résoudre $f(\boldsymbol{x}') \leq \overline{f}$ grâce à un développement de Taylor d'ordre 2
appliquer quelques itérations de Newton à $\boldsymbol{G}$
couper $\boldsymbol{x}$ en deux : $\boldsymbol{x}_1$ et $\boldsymbol{x}_2$
évaluer $f(\boldsymbol{x}_1)$: $[\underline{f_1}, \overline{f_1}]$
mettre à jour $\overline{f}$: si $\overline{f_1} \leq \overline{f}$ alors $\overline{f} := \overline{f_1}$
si $\underline{f_1} > \overline{f}$ ou si $\boldsymbol{G}(\boldsymbol{x}_1) \not\ni 0$ ou si $\boldsymbol{H}(\boldsymbol{x}_1)$ ne vérifie pas la condition de convexité alors
jeter $\boldsymbol{x}_1$ // $\boldsymbol{x}_1$ ne peut pas contenir le minimum x^*
sinon si $w(\boldsymbol{x}_1) \leq \varepsilon$ alors
ranger $\boldsymbol{x}_1$ dans Res
sinon
ranger $\boldsymbol{x}_1$ dans $\mathcal{L}$
idem avec $\boldsymbol{x_2}$

Fin :
mettre à jour l'encadrement de $f(x^*) = [\underline{f}, \overline{f}]$:
$\underline{f} = \min_{y=f(\boldsymbol{x}), x \in Res} \underline{y}$, $\overline{f} = \max_{y=f(\boldsymbol{x}), x \in Res} \overline{y}$
éliminer les $\boldsymbol{x}$ de Res tels que $\underline{y} > \overline{f}$ avec $[\underline{y}, \overline{y}] = f(\boldsymbol{x})$

3.3 Force : informations globales, optimisation globale sous contraintes

Fréquemment, les problèmes d'optimisation globale sont des problèmes sous contraintes : on cherche le minimum d'une fonction f qui vérifie en outre un certain nombre d'équations et inéquations. Mathématiquement, le problème s'écrit :

$$(\mathcal{P}_c) \begin{cases} \min_x f(x) & & & \\ \text{contraintes :} & p_i(x) & \leq & 0 \quad 1 \leq i \leq m \\ & q_j(x) & = & 0 \quad 1 \leq j \leq r \end{cases}$$

La difficulté est maintenant concentrée en deux points : déterminer des points qui satisfont les contraintes et trouver leur minimum x^*, qui lui ne vérifie plus grad $f(x^*) = 0$ ni f localement convexe en x^*.

Deux approches (au moins) sont possibles et complémentaires. La première est classique en mathématiques et analyse numérique : elle consiste à transformer $\mathcal{P}_c$ en un problème d'optimisation globale sans contraintes. Pour cela, les contraintes sont intégrées à la fonction à minimiser, sous forme de pénalités, ou alors en

remplaçant dans Newton la fonction grad f par

$$\lambda_0 \text{ grad } f + \sum_i \lambda_i \text{ grad } p_i + \sum_j \mu_j \text{ grad } q_j$$

où les λ_i et μ_j sont appelés multiplicateurs de Lagrange. Ces nouvelles variables doivent vérifier certaines conditions appelées conditions de Kuhn et Tucker si $\lambda_0 = 1$ et de Fritz - John dans le cas général. Ces conditions sont des égalités à 0 et peuvent dont être traitées par l'algorithme de Newton. Nous renvoyons à [16] pour une description plus précise et une discussion plus détaillée de cette méthode, qui inclut la détermination d'un pavé contenant les paramètres λ_i et μ_j, qui sera fourni en entrée de l'algorithme de Newton. La seconde approche utilise les contraintes pour rejeter ou réduire le pavé de recherche à chaque étape. Un pavé $\boldsymbol{x}$ sera rejeté s'il ne satisfait pas les contraintes, par exemple s'il existe un indice i tel que $p_i(\boldsymbol{x}) > 0$ ou $q_i(\boldsymbol{x}) \not\ni 0$. Si ce pavé $\boldsymbol{x}$ n'est pas rejeté, on tentera d'éliminer de $\boldsymbol{x}$ les parties ne satisfaisant pas les contraintes. Pour cela, les techniques présentées au §7 ou dans [1] peuvent être mises en œuvre.

On peut n'utiliser qu'une seule de ces approches : dans le second cas, on supprime les appels à Newton pour résoudre grad $f = 0$ et le test de convexité. On peut aussi combiner les deux approches pour réduire à chaque étape le pavé courant autant que possible. Par exemple, un algorithme d'optimisation globale sous contraintes peut s'écrire de la façon suivante.

Données : f et $\boldsymbol{f}$ une extension intervalle de f,
$\boldsymbol{x}_0$ le pavé de recherche,
$\boldsymbol{p}_i$, $1 \leq i \leq m$, des extensions intervalles des p_i définissant les contraintes de type inégalité,
$\boldsymbol{q}_j$, $1 \leq j \leq r$, des extensions intervalles des q_j définissant les contraintes d'égalité à 0
Résultats : $[\underline{f};\, \overline{f}] \ni f^* = \min_{x \in \boldsymbol{x}_0} f(x)$
Res une liste de pavés $\boldsymbol{x}^*$ tels que $\boldsymbol{f}(\boldsymbol{x}^*) \cap [\underline{f};\, \overline{f}] \neq \emptyset$
pour certains éléments de Res, un certificat d'existence d'un point satisfaisant les contraintes
Initialisations :
$\mathcal{L} := \{\boldsymbol{x}_0\}$ la liste des pavés en attente de traitement
Res $:= \emptyset$
$[\underline{f};\, \overline{f}] := \boldsymbol{f}(\boldsymbol{x}_0)$
tant que $\mathcal{L} \neq \emptyset$ **faire**
sortir un pavé $\boldsymbol{x}$ de $\mathcal{L}$
$\boldsymbol{x}' :=$ réduire $\boldsymbol{x}$ // en utilisant les contraintes aussi : cf. ci-dessous

résoudre $\boldsymbol{f}(\boldsymbol{x}') \leq \overline{f}$
appliquer quelques pas de Newton à λ_0 grad $\boldsymbol{f} + \sum_i \lambda_i$ grad $\boldsymbol{p}_i + \sum_j$ grad $\boldsymbol{q}_j$
appliquer quelques pas de l'algorithme de propagation et rétro-propagation
s'il est prouvé que $\boldsymbol{x}'$ contient un point satisfaisant les contraintes et si $\boldsymbol{f}(\boldsymbol{x}') < \overline{f}$ alors
$\overline{f} := \max \boldsymbol{f}(\boldsymbol{x}')$
éliminer de Res les pavés $\boldsymbol{x}^*$ pour lesquels $\boldsymbol{f}(\boldsymbol{x}^*) > \overline{f}$
si $\boldsymbol{x}'$ satisfait le critère d'arrêt alors
ranger $\boldsymbol{x}'$ dans Res
sinon
$(\boldsymbol{x}_1, \boldsymbol{x}_2) :=$ split$(\boldsymbol{x}')$ (bissection de $\boldsymbol{x}'$)
si $\boldsymbol{f}(\boldsymbol{x}_1) > \overline{f}$ ou s'il existe i tel que $\boldsymbol{p}_i(\boldsymbol{x}') > 0$ ou $\boldsymbol{q}_i(\boldsymbol{x}') \not\ni 0$ alors
éliminer $\boldsymbol{x}_1$
sinon
ranger $\boldsymbol{x}_1$ dans $\mathcal{L}$
idem pour $\boldsymbol{x}_2$
Fin : $\underline{f} := \min_{\boldsymbol{x} \in Res} \underline{\boldsymbol{f}(\boldsymbol{x})}$, $\overline{f} := \min_{\boldsymbol{x} \in Res} \overline{\boldsymbol{f}(\boldsymbol{x})}$
le min et le max étant pris sur les pavés pour lesquels il est prouvé qu'ils contiennent un point satisfaisant les contraintes.

3.4 Faiblesse : le résultat dépend de l'expression utilisée

À partir des opérations et fonctions définies sur des intervalles au §, on peut étendre pour les intervalles toute fonction, dès qu'elle est définie par une expression ne faisant intervenir que ces calculs. Soit f une fonction de $D \subset \mathbb{R}^m$ dans $\mathbb{R}^n$, la propriété fondamentale de garantie des résultats permet d'assurer que pour tout pavé $\boldsymbol{x} \subset D$, le pavé obtenu en substituant les variables scalaires par les composantes intervalles correspondantes de $\boldsymbol{x}$ est un surencadrement de l'image de $\boldsymbol{x}$ par f. L'idéal serait de déterminer le plus petit pavé contenant $f(\boldsymbol{x})\ldots$

Commençons par examiner le cas des fonctions polynomiales à une seule variable. Si $f : \mathbb{R} \to \mathbb{R}$ est définie par $x \mapsto x^2 - 2x + 1$ et si $\boldsymbol{x} = [-1; 3]$, en remplaçant x par $\boldsymbol{x}$ dans l'expression de f, on obtient $\boldsymbol{x}^2 - 2\boldsymbol{x} + 1 = [-5; 12]$. Si on utilise plutôt l'expression — équivalente en arithmétique réelle — $f(x) = x(x-2)+1$, on obtient $\boldsymbol{x}(\boldsymbol{x}-2)+1 = [-8; 4]$ et enfin en utilisant l'écriture factorisée $f(x) = (x-1)^2$ on obtient $(\boldsymbol{x}-1)^2 = [0; 4] = f(\boldsymbol{x})$. Cet exemple illustre clairement le fait que des expressions équivalentes en arithmétique réelle ne le sont plus en arithmétique par intervalles, même si chacune donne lieu à un surencadrement de

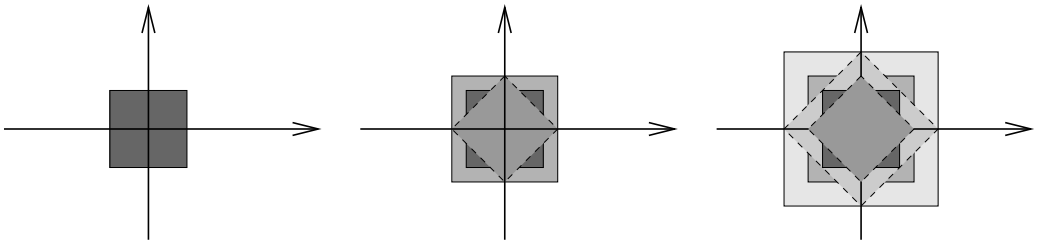


FIG. 9 – Effet enveloppant : après deux rotations successives de $\pi/4$ du petit carré central, on ne retrouve pas le petit carré central mais le grand carré.

l'image de $\boldsymbol{x}$ par f.

Nous venons de mettre le doigt sur l'inconvénient majeur de l'arithmétique par intervalles, à savoir la surestimation (ou encadrement trop large) des résultats.

Deux phénomènes ont été identifiés comme explicatifs de ce problème. Le premier problème, connu sous le nom d'"effet enveloppant" ou *wrapping effect*, est illustré par la figure 9 : si f est une fonction de $\mathbb{R}^m$ dans $\mathbb{R}^n$, l'image par f d'un pavé $\boldsymbol{x}$ de $\mathbb{R}^m$ sera un ensemble de $\mathbb{R}^n$ de forme quelconque. Or l'arithmétique par intervalles impose que le résultat calculé soit un pavé de côtés parallèles aux axes qui contient $f(\boldsymbol{x})$ et ce pavé peut être de volume beaucoup plus grand que celui de $f(\boldsymbol{x})$.

Le second phénomène expliquant la surestimation des résultats a été observé dans les exemples précédents, il s'agit du problème de dépendance des données (*data dependency*) ou décorrélation des données : lors du remplacement du calcul de $x * x$ par $\boldsymbol{x} * \boldsymbol{x}$, le résultat est $\{x * y \mid x \in \boldsymbol{x},\ y \in \boldsymbol{x}\}$, autrement dit l'égalité entre x et y est perdue. De même, dans l'énoncé de la sous-distributivité, l'écriture $\boldsymbol{x} * \boldsymbol{y} + \boldsymbol{x} * \boldsymbol{z}$ est la traduction de $\{x*y+x'*z \,|\, x \in \boldsymbol{x}, x' \in \boldsymbol{x}, y \in \boldsymbol{y}, z \in \boldsymbol{z}\}$ et ne tient pas compte de l'identité de x et x', c'est pour cette raison que cet intervalle est plus large que $\boldsymbol{x} * (\boldsymbol{y} + \boldsymbol{z})$ dans lequel la variable $\boldsymbol{x}$ n'apparaît qu'une seule fois et ne peut donc pas être décorrélée de ses autres occurrences.

Cette remarque est à la base du théorème suivant, dû à Moore [31].
Théorème. Si f est une fonction continue de $\mathbb{R}^n$ dans $\mathbb{R}$ définie par une expression dans laquelle chaque variable x_i apparaît au plus une fois, alors pour tout pavé $\boldsymbol{x} \subset \mathbb{R}^n$, l'évaluation par intervalles obtenue en remplaçant x_i par la composante correspondante de $\boldsymbol{x}$ est exactement égale à $f(\boldsymbol{x})$.

3.5 Faiblesse : tous les problèmes (ou presque) sont NP-durs

On vient de voir que le résultat d'un calcul dépend de l'expression utilisée. Or on souhaite obtenir un résultat précis, et dans l'idéal le plus petit résultat possible, c'est-à-dire le plus petit intervalle (au sens de l'inclusion) qui contient le résultat exact, ce dernier pouvant être un ensemble de forme quelconque. Cet espoir est une utopie dans la majorité des cas, puisque beaucoup de problèmes sont NP-durs. Nous allons établir une liste d'exemples, pour montrer l'ampleur de la classe des problèmes NP-durs en arithmétique par intervalles.

Un premier théorème, dû à Gaganov [13], établit que le problème de l'évaluation à ε près d'une fonction polynomiale à plusieurs variables et à coefficients rationnels, donc l'un des problèmes apparemment les plus simples que l'on puisse envisager, est NP-dur. Pour une fonction quelconque, l'évaluation optimale de cette fonction sur un intervalle est donc un but inaccessible.

Pour ce qui touche à l'algèbre linéaire, J. Rohn et ses co-auteurs ont établi l'appartenance à la classe des problèmes NP-durs de bon nombre de problèmes usuels. En voici une liste non exhaustive et en vrac :

- déterminer si une matrice intervalle est régulière, c'est-à-dire si toutes les matrices ponctuelles qu'elle contient sont inversibles [45] ;
- déterminer si l'ensemble solution de $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ (au sens défini au §) est borné [42] ;
- calculer un encadrement optimal de cet ensemble solution [46] ;
- calculer un encadrement à $1/4n^4$ près de l'ensemble solution du système linéaire $n \times n$ $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ [44] ;
- calculer la norme d'une matrice $\boldsymbol{A}$ [43] ;
- déterminer la factorisation LU, à ε près, d'une matrice $\boldsymbol{A}$ par l'algorithme d'élimination de Gauss avec pivot partiel [43] ;
- déterminer si un système linéaire rectangulaire $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ admet une solution strictement positive [45] ;
- déterminer s'il existe un système linéaire ponctuel $Ax = b$ avec $A \in \boldsymbol{A}$ et $b \in \boldsymbol{b}$, rectangulaire, qui admet une solution [43] ;
- déterminer l'encadrement optimal de l'ensemble solution du programme linéaire

$$\begin{cases} \min \boldsymbol{c}^t.\boldsymbol{x} & \\ \text{sous les contraintes} & \boldsymbol{A}.\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq 0 \end{cases}$$

  ici le calcul de la borne inférieure est dans P et celui de la borne supérieure est NP-dur [45].

– ...

On vient de le voir, une très grande partie des problèmes usuellement traités sont NP-durs. Cela justifie le fait que la complexité des problèmes étudiés dans la suite de ce cours ne sera jamais mentionnée, sauf exception.

Comme on peut le constater avec les premiers algorithmes vus ici, d'optimisation globale, et comme cela se confirmera par la suite, le seul argument permettant d'établir facilement la terminaison des algorithmes est le fait que les intervalles traités sont coupés en deux à chaque étape et qu'ils finiront donc par avoir une largeur inférieure à ε, si on prend soin de les couper selon leur plus grand côté. Cet argument conduit à une complexité exponentielle des algorithmes proposés (ce qui rejoint le caractère NP-dur des problèmes). En pratique, on cherche des procédures de réduction les plus efficaces possibles afin de ne couper les pavés en deux qu'en dernier recours, même si les étudier de façon théorique pour affiner les bornes de complexité reste une tâche ardue.

4 Évaluation d'une expression : développements de Taylor

La partie précédente a mis en évidence les avantages uniques de l'arithmétique par intervalles, au travers des algorithmes pour l'optimisation globale d'une fonction continue, mais également ses faiblesses, l'une d'elles étant la dépendance vis-à-vis de l'expression utilisée pour le calcul.

Déterminer l'expression qui conduira au meilleur résultat est un problème difficile et on ne sait pas encore très bien comment l'aborder. En revanche, à partir d'une expression donnée, on peut montrer que pour des intervalles pas trop larges, l'utilisation de développements de Taylor d'ordres 1 et 2 réduit le phénomène de surestimation. Le principe fondamental consiste à utiliser une expression qui fait apparaître le plus de calculs scalaires possibles et peu de calculs par intervalles. C'est ce qui est présenté au §4.2. Mais commençons par définir l'extension d'une fonction, ou plutôt d'une expression, à des entrées intervalles et les propriétés désirables d'une telle extension. La suite de cette section est reprise intégralement de [39].

4.1 Évaluation des fonctions : fonctions d'inclusion naturelles

Si on désire évaluer l'image d'une fonction à variables scalaires sur un vecteur d'intervalles, on définit une extension intervalle $\boldsymbol{f}$ de f à partir d'une expression définissant f. Cette fonction $\boldsymbol{f}$ est appelée *fonction d'inclusion naturelle* de f. En particulier, cette extension $\boldsymbol{f}$ vérifie[11] que pour tout intervalle ponctuel x (avec l'abus de notation $x = \{x\}$ pour les points), $\boldsymbol{f}(x) = f(x)$. Une fonction d'inclusion naturelle est également monotone pour l'inclusion, c'est-à-dire que :

$$\forall \boldsymbol{x},\ \forall \boldsymbol{y},\ \text{si } \boldsymbol{x} \subset \boldsymbol{y} \text{ alors } \boldsymbol{f}(\boldsymbol{x}) \subset \boldsymbol{f}(\boldsymbol{y}).$$

Cette propriété semble aller de soi, nous verrons au §4.2 que certaines méthodes d'évaluation ne la vérifient pas. On peut en effet définir une fonction d'inclusion ou *extension intervalle* $\boldsymbol{f}$ d'une fonction scalaire $f : D \subset \mathbb{R}^n \to \mathbb{R}$ par la simple propriété d'inclusion

$$\forall \boldsymbol{x} \in \mathbb{IR}^n,\ \boldsymbol{x} \subset D,\ f(\boldsymbol{x}) \subset \boldsymbol{f}(\boldsymbol{x}).$$

Si f est une fonction à valeurs dans $\mathbb{R}^n$, l'étude d'une fonction d'inclusion pour f revient à l'étude d'une fonction d'inclusion pour chacune des composantes de f ; c'est pour cette raison que nous nous concentrerons sur l'étude des fonctions à valeurs réelles.

Afin de pouvoir comparer la qualité de deux évaluations par intervalles, une distance est nécessaire. On peut munir $\mathbb{IR}$ d'une structure d'espace métrique grâce à la distance (de Hausdorff) q définie par :

$$\begin{aligned} q(\boldsymbol{x}, \boldsymbol{y}) &= \min\{q \in \mathbb{R}^+ \mid \boldsymbol{x} \subset \boldsymbol{y} + [-q; q] \text{ et } \boldsymbol{y} \subset \boldsymbol{x} + [-q; q]\} \\ &= \max(|\underline{x} - \underline{y}|, |\overline{x} - \overline{y}|) \\ &= |\check{x} - \check{y}| + 1/2|w(\boldsymbol{x}) - w(\boldsymbol{y})|. \end{aligned}$$

Comme très souvent $\boldsymbol{x}$ sera l'intervalle minimal et $\boldsymbol{y}$ sera une surestimation de $\boldsymbol{x}$: $\boldsymbol{y} \supset \boldsymbol{x}$, cette distance $q(\boldsymbol{x}, \boldsymbol{y}) = \max(|\underline{x} - \underline{y}|, |\overline{x} - \overline{y}|)$ mesurera de combien $\boldsymbol{y}$ " déborde " de chaque côté de $\boldsymbol{x}$.
La distance classiquement utilisée dans $\mathbb{IR}^n$ est le maximum des distances pour chacune des composantes.

Un premier théorème encourageant a été donné par Moore dans [31], on peut également le trouver dans [33] : sous certaines conditions assez classiques sur le

[11] Cette égalité est vraie tant que le calcul est exact. Une implantation basée sur l'arithmétique flottante ne vérifiera pas cette égalité puisque, à cause de la prise en compte des erreurs d'arrondi, $\boldsymbol{f}(\{x\})$ sera un intervalle non ponctuel dans le cas général.

caractère lipschitzien de l'expression à évaluer, l'écart entre l'intervalle calculé et l'image de la fonction correspondante est proportionnel au diamètre de l'intervalle d'entrée.
Théorème. Soit f une fonction en n variables définie par une expression arithmétique, lipschitzienne au sens intervalle en $\boldsymbol{x}_0 \in I\!I\!R^n$, on a :

$$q(f(\boldsymbol{x}), \boldsymbol{f}(\boldsymbol{x})) = \mathcal{O}(w(\boldsymbol{x})).$$

Si on désire un encadrement plus fin de l'image de $\boldsymbol{x}$ par f, une première solution — qui peut être considérée comme naïve tant que l'on dispose de techniques plus efficaces mais sera souvent le dernier recours, cf. §5.2 et 3.2 — consiste à découper le pavé $\boldsymbol{x}$ en petits sous-pavés $\boldsymbol{x}_i$. L'image de $\boldsymbol{x}$ par f, $f(\boldsymbol{x})$, est incluse dans l'union des $\boldsymbol{f}(\boldsymbol{x}_i)$ et, si les $\boldsymbol{x}_i$ ont une largeur assez petite, on a pour chaque $\boldsymbol{x}_i$ une surestimation inférieure à ε, d'où finalement $q(f(\boldsymbol{x}), \bigcup_i \boldsymbol{f}(\boldsymbol{x}_i)) \leq \varepsilon$. Cependant une telle stratégie pour une fonction f à n variables implique une découpe de $\boldsymbol{x}$ en un nombre de sous-pavés exponentiel en le nombre de variables. Cela rejoint le théorème de Gaganov sur la difficulté d'évaluer précisément une fonction, cf. §4.

D'autres solutions reposent sur l'utilisation des formules de Taylor-Lagrange (l'utilisation des fonctions pente ne sera pas abordée dans ce cours) et surtout sur la constatation suivante : toute formule dans laquelle on calcule le plus possible avec des scalaires et le moins possible avec des intervalles conduit à de bons encadrements des résultats cherchés. Les fonctions d'inclusion correspondantes sont présentées au paragraphe suivant.

4.2 Évaluation de fonctions : formes centrées et formules de Taylor-Lagrange

Soit f une fonction de $I\!R^n$ dans $I\!R$ continue et différentiable, supposons que l'on cherche à encadrer $f(\boldsymbol{x})$ pour un pavé $\boldsymbol{x} \subset I\!R^n$. La formule de Taylor-Lagrange au premier ordre (également appelée théorème des accroissements finis) indique que pour tout $x \in \boldsymbol{x}$ et tout $y \in \boldsymbol{x}$,

$$\exists \xi \in]x; y[\text{ (ou }]y; x[\text{ si } y < x) \ / \ f(y) = f(x) + f'(\xi).(y - x).$$

À de rares exceptions près, la valeur de ξ est inconnue ; cependant un encadrement de $f'(\xi)$ est donné par $f'(]x; y[)$ (ou $f'(]y; x[)$ si $y < x$, mais par abus de notation nous le noterons aussi $f(]x; y[)$), si de plus on dispose d'une fonction d'inclusion

$\boldsymbol{f}'$ pour f', on a l'inclusion suivante :

$$f(y) \in f(x) + f'(]x; y[).(y - z) \subset f(x) + \boldsymbol{f}'([x; y]).(y - x).$$

Comme ceci est vrai pour tout $x \in \boldsymbol{x}$ et pour tout $y \in \boldsymbol{x}$, on a pour $x \in \boldsymbol{x}$ fixé

$$f(\boldsymbol{x}) \subset f(x) + \boldsymbol{f}'(\boldsymbol{x}).(\boldsymbol{x} - x)$$

et finalement la fonction $\boldsymbol{f}_{TL1}$ définie par $\boldsymbol{f}_{TL1}(\boldsymbol{x}) = f(x) + \boldsymbol{f}'(\boldsymbol{x}).(\boldsymbol{x} - x)$ est une fonction d'inclusion pour f sur $\boldsymbol{x}$. Le choix de $x \in \boldsymbol{x}$ est arbitraire ; ce point s'appelle *centre* mais n'est pas nécessairement le milieu de $\boldsymbol{x}$ et la forme correspondante $\boldsymbol{f}_{TL1}$ est une *forme centrée*. Ce n'est pas une expression arithmétique puisqu'elle fait intervenir un point de l'intervalle $\boldsymbol{x}$ et qu'un point n'est pas une quantité accessible par les opérations et fonctions élémentaires par intervalles.

Le théorème suivant permet de majorer le surencadrement et de se persuader qu'une forme centrée est intéressante dès lors que l'intervalle $\boldsymbol{x}$ est petit.
Théorème [33]. Si la fonction d'inclusion $\boldsymbol{f}'$ de f' est lipschitzienne au sens intervalle, alors la fonction d'inclusion donnée par la formule de Taylor-Lagrange au premier ordre vérifie :

$$q(\boldsymbol{f}_{TL1}(\boldsymbol{x}), f(\boldsymbol{x})) = \mathcal{O}(w(\boldsymbol{x})^2).$$

On a donc une propriété d'*approximation quadratique* en utilisant comme fonction d'inclusion la formule de Taylor-Lagrange d'ordre 1. Une telle propriété signifie que si $\boldsymbol{x}$ est petit, alors cette formule donne de meilleurs résultats qu'une fonction d'inclusion naturelle pour laquelle l'approximation est seulement linéaire ; en revanche lorsque $\boldsymbol{x}$ est large, la fonction d'inclusion naturelle fournira un meilleur surencadrement que la forme centrée. La détermination de la largeur seuil w_0 à partir de laquelle l'une des deux formules est meilleure que l'autre est un problème non résolu. Une solution pratique consiste à évaluer $f(\boldsymbol{x})$ à l'aide d'une fonction d'inclusion naturelle et d'une forme centrée et à prendre l'intersection des deux encadrements obtenus, cependant cette solution est coûteuse en nombre d'opérations.

Revenons au choix du point x par rapport auquel on effectue le développement de Taylor : il est possible de choisir x de façon à maximiser la borne gauche du résultat, ou bien à minimiser la borne droite, ou bien à minimiser la largeur de l'évaluation [5]. Le choix qui conduit à une largeur minimale consiste à prendre pour x le milieu de l'intervalle $\boldsymbol{x}$. Ce choix présente de plus l'avantage de fournir une fonction d'inclusion monotone pour l'inclusion, comme le précise un théorème dû à Caprani et Madsen [10]. Ce n'est plus vrai si le point utilisé pour

le développement n'est pas le milieu de l'intervalle.

On conserve également la monotonie de l'inclusion lorsque l'on évalue $f(\boldsymbol{x})$ en prenant l'intersection de l'évaluation à l'aide d'une fonction d'inclusion naturelle et du développement de Taylor-Lagrange à l'ordre 1 en mid$(\boldsymbol{x})$, puisque chacune des deux fonctions d'inclusion l'est.

Une idée naturelle consiste à s'intéresser aux développements de Taylor avec restes de Lagrange d'ordres plus élevés, dans l'espoir d'obtenir des approximations d'ordre supérieur à 2. Une première remarque est qu'il n'est pas possible d'atteindre un ordre strictement supérieur à 2 si l'ensemble des opérations algébriques ne contient que $+$, $-$, $*$ et $/$: l'opération manquante, celle sans laquelle l'ordre 3 est inaccessible, est l'élévation au carré [18]. Par bonheur elle était incluse dans notre boîte à outils d'opérations et de fonctions, cf. §2.2, mais cela peut ne pas suffire... Une seconde constatation est que, pour des fonctions à n variables, il devient rapidement difficile d'exprimer les différentielles d'ordre élevé, ce qui explique qu'en pratique on n'utilise que des développements de Taylor d'ordres 1 ou 2. Enfin, pour ces mêmes fonctions multivariées, il est difficile d'exprimer le développement de Taylor à l'ordre 2 en n'utilisant que des élévations au carré, ce qui empêche d'obtenir une approximation cubique.
En revanche, l'obtention des premières dérivées partielles est maintenant bien maîtrisée, grâce aux progrès de la différentiation automatique [14]. Cependant, les logiciels présentés au §2.4 ne les calculent pas tous. Même parmi les logiciels implantant la différentiation automatique, la qualité des valeurs calculées dépend du mécanisme choisi : la différentiation directe (ou *forward*) est la plus simple à mettre en œuvre tandis que la différentiation automatique régressive (ou *backward*) donne de meilleurs résultats. C'est pour cela que la description sommaire des logiciels mentionnait ces informations.

Cette section était consacrée à l'évaluation par intervalles d'une expression et à quelques techniques qui limitent le phénomène de surestimation, au moins pour des intervalles de faible largeur. La suite de ce cours est consacrée au détail des algorithmes évoqués aux §3.2 et 3.3, c'est-à-dire aux briques de base des algorithmes d'optimisation globale. La première de ces briques de base est l'algorithme de Newton par intervalles.

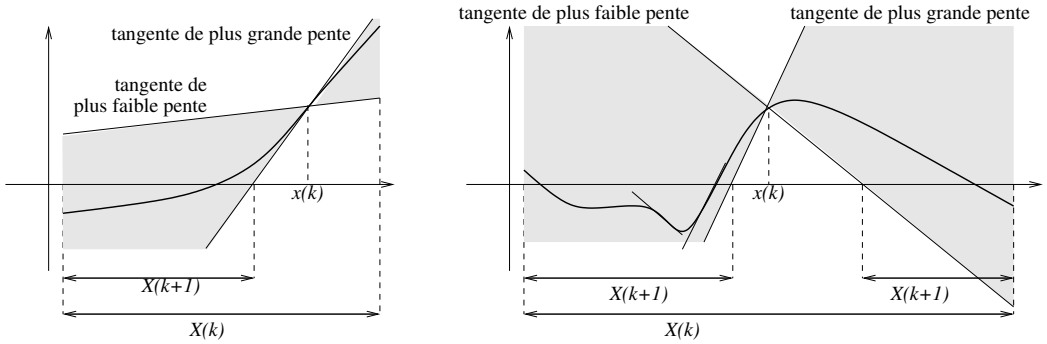


FIG. 10 – Schéma de la méthode de Newton dans le cas à une seule variable.

5 Recherche des zéros d'une fonction : Newton par intervalles

L'algorithme de Newton par intervalles permet de localiser les zéros d'une fonction de classe $\mathcal{C}^1$. Rappelons que dans le cas d'une fonction $f : \mathbb{R} \to \mathbb{R}$, l'itération de Newton est définie comme suite : si x^k est l'itéré courant, le nouvel itéré x^{k+1} est donné par

$$x^{k+1} = x^k - f(x^k)/f'(x^k).$$

Cette formule ne peut pas être transposée telle quelle au calcul par intervalles. En effet, si l'on calcule la largeur de x^{k+1}, on obtient

$$\begin{array}{rcl} w(x^{k+1}) & = & w(x^k) + w\left(f(x^k)/f'(x^k)\right) \\ & \geq & w(x^k), \end{array}$$

autrement dit cette itération diverge.

5.1 Présentation de la méthode de Newton par intervalles : cas univarié

Pour avoir convergence, il faut donc déterminer une itération contractante. Pour cela, il faut appliquer le principe général déjà énoncé : déterminer une formule comportant un maximum de calculs scalaires et un minimum de calculs sur les intervalles. C'est possible pour l'itération de Newton et cela s'illustre aisément par la figure 10.

On cherche les zéros de la fonction f, c'est-à-dire les points d'intersection de la courbe représentative de f avec l'axe des abscisses. Comme il s'agit de calcul par

intervalles, il faut garantir de ne pas oublier un seul de ces zéros. Pour cela, la courbe représentative de f est remplacée par le cône grisé sur la figure. Ce cône est défini par le point[12] $(x, f(x))$ avec x une abscisse quelconque appartenant à l'intervalle considéré $\boldsymbol{x}$ et ses pentes sont données par les extrêma des dérivées de f sur $\boldsymbol{x}$. En effet, la formule de Taylor-Lagrange indique que pour tout $x' \in \boldsymbol{x}$, $\exists \xi \in [x, x']$ et *a fortiori* $\exists \xi \in \boldsymbol{x}$ vérifiant

$$f(x') = f(x) + f'(\xi).(x' - x) \in f(x) + f'(\boldsymbol{x})(\boldsymbol{x} - x).$$

Il suffit donc d'avoir un intervalle contenant $f'(\boldsymbol{x})$, qui peut être obtenu en évaluant une expression pour f' sur $\boldsymbol{x}$, pour pouvoir déterminer le cône grisé de la figure. Cette expression pour f' peut éventuellement être donnée par un mécanisme de différentiation automatique de f.

L'ensemble des zéros de f, c'est-à-dire des points d'intersection de la courbe représentative de f avec l'axe des abscisses, est inclus dans l'intersection de ce cône grisé avec l'axe des abscisses et cette intersection est facile à déterminer. En effet, tout point (x', y') du cône se trouve sur la droite passant par $(x, f(x))$ et par lui-même et l'équation de cette droite est $y' - f(x) = \alpha(x' - x)$ avec $\alpha \in f'(\boldsymbol{x})$. En particulier, tout point du cône qui se trouve sur l'axe des abscisses vérifie à la fois cette équation et $y' = 0$. On obtient donc que l'abscisse x' d'un tel point vérifie

$$-f(x) = \alpha(x' - x) \text{ avec } \alpha \in f'(\boldsymbol{x})$$

ou encore

$$x' = x - \frac{f(x)}{\alpha}.$$

L'ensemble des points d'intersection du cône grisé avec l'axe des abscisses est donc donné par

$$x - f(x)/f'(\boldsymbol{x}).$$

Cette formule ressemble à celle de l'itération de Newton. On note cependant qu'un seul intervalle intervient, comme argument de f'.

L'algorithme de Newton par intervalles dans le cas d'une seule variable est donné ci-dessous.

Données : $\boldsymbol{f}$ une extension intervalle de f et $\boldsymbol{f}'$ une extension intervalle de f'
$\boldsymbol{x}_0$ un pavé de recherche
Résultat : Res une liste de pavés susceptibles de contenir un zéro de f

[12] Dans le cas d'une implantation avec une arithmétique approchée, le point $(x, f(x))$ devra être remplacé par le segment $(x, \boldsymbol{f}(x))$ puisque, à cause des erreurs d'arrondi, l'évaluation de $f(x)$ produira un intervalle.

Initialisations :

$\mathcal{L} := \{\boldsymbol{x}_0\}$ la liste des pavés en attente de traitement

Res $:= \emptyset$

tant que $\mathcal{L} \neq \emptyset$ **faire**

sortir un pavé $\boldsymbol{x}$ de $\mathcal{L}$

choisir x un point quelconque de $\boldsymbol{x}$, par exemple $x = \text{mid}(\boldsymbol{x})$

$N(x, \boldsymbol{x}) = x - \boldsymbol{f}(x)/\boldsymbol{f}'(\boldsymbol{x})$

$\boldsymbol{x}' = N(x, \boldsymbol{x}) \cap \boldsymbol{x}$

si $\boldsymbol{x}' \neq \emptyset$ et $\boldsymbol{f}(\boldsymbol{x}') \ni 0$ alors

ranger $\boldsymbol{x}'$ dans $\mathcal{L}$ ou Res

Dans l'algorithme qui précède, deux points sont à commenter. Tout d'abord, on prend pour nouvel itéré l'intersection de l'intervalle calculé avec le précédent itéré, pour limiter la recherche à l'intervalle $\boldsymbol{x}_0$ initialement considéré. On voit en effet sur la figure que, sans cette intersection, le nouvel itéré pourrait déborder du cadre prescrit. C'est ce qui rend l'itération contractante. Ensuite, on utilise à nouveau une structure de liste pour les intervalles en attente de traitement, même si on part d'un seul intervalle. On voit sur la partie droite de la figure que dans le cas où f' s'annule sur $\boldsymbol{x}$, l'intersection du cône grisé avec l'axe des abscisses est constitué de deux parties disjointes et donc $N(x, \boldsymbol{x})$ est composé de deux intervalles. Dans ce dernier cas, comme $\boldsymbol{f}'(\boldsymbol{x})$ contient 0, la division n'est pas celle définie au §2.2 et il s'agit d'une division dite étendue [21, 38], définie comme suit : si $\boldsymbol{x} = [\underline{x}, \overline{x}]$ et $\boldsymbol{y} = [\underline{y}, \overline{y}]$,

$$\frac{\boldsymbol{x}}{\boldsymbol{y}} = \begin{cases} \left[\overline{x}/\underline{y}, +\infty\right] & \text{si } \overline{x} \leq 0 \text{ et } \overline{y} = 0 \\ [-\infty, \overline{x}/\overline{y}] \cup \left[\overline{x}/\underline{y}, +\infty\right] & \text{si } \overline{x} \leq 0 \text{ et } \underline{y} < 0 < \overline{y} \\ [-\infty, \overline{x}/\overline{y}] & \text{si } \overline{x} \leq 0 \text{ et } \underline{y} = 0 \\ [-\infty, +\infty] & \text{si } \underline{x} < 0 < \overline{x} \\ \left[-\infty, \underline{x}/\underline{y}\right] & \text{si } \underline{x} \geq 0 \text{ et } \overline{y} = 0 \\ \left[-\infty, \underline{x}/\underline{y}\right] \cup [\underline{x}/\overline{y}, +\infty] & \text{si } \underline{x} \geq 0 \text{ et } \underline{y} < 0 < \overline{y} \\ [\underline{x}/\overline{y}, +\infty] & \text{si } \underline{x} \geq 0 \text{ et } \underline{y} = 0 \end{cases}$$

En pratique, quand on implante l'algorithme de Newton, il reste à préciser le test d'arrêt et également, pour prouver la terminaison de l'algorithme, à garantir que l'itération est suffisamment contractante. Un test d'arrêt couramment utilisé consiste à exiger que la largeur de $\boldsymbol{x}$ soit inférieure à un seuil ε_x, autrement dit que $\boldsymbol{x}$ soit un encadrement précis d'un zéro de f, ou alors que la largeur de $\boldsymbol{f}(\boldsymbol{x})$ soit suffisamment faible, $|\boldsymbol{f}(\boldsymbol{x})| \leq \varepsilon_y$: dans ce cas, f est considérée comme suffisamment proche de 0. Pour garantir que l'itération est suffisamment contractante, on coupe éventuellement l'intervalle $\boldsymbol{x}$ en deux si le nouvel itéré $\boldsymbol{x}'$ n'est pas considéré comme suffisamment réduit.

5.2 Présentation de la méthode de Newton par intervalles : cas multivarié

Dans le cas où f est une fonction à plusieurs variables $f : \mathbb{R}^n \to \mathbb{R}^n$, la dérivée est remplacée par la Jacobienne de f et l'itération de Newton devient :

$$\begin{array}{l} \boldsymbol{x}^0 \text{ donné} \\ \boldsymbol{x}^{k+1} = \left(\tilde{x} - \boldsymbol{f}'(\boldsymbol{x}^k)^{-1}.f(\tilde{x})\right) \cap \boldsymbol{x}^k. \end{array}$$

Plus précisément, on détermine une solution $N(\tilde{x}, \boldsymbol{x}^k)$ du système linéaire $\boldsymbol{f}'(\boldsymbol{x}^k)(\tilde{x} - N(\tilde{x}, \boldsymbol{x}^k)) = f(\tilde{x})$ et le nouvel itéré est $\boldsymbol{x}^{k+1} = N(\tilde{x}, \boldsymbol{x}^k) \cap \boldsymbol{x}^k$. Ici encore, l'itération ainsi construite est clairement monotone décroissante pour l'inclusion puisque par construction $\boldsymbol{x}^{k+1} \subset \boldsymbol{x}^k$.

5.2.1 Cas où la fonction admet au plus un zéro

La mise en œuvre de l'itération de Newton nécessite la résolution d'un système linéaire dont la matrice est $\boldsymbol{f}'(\boldsymbol{x})$. On supposera pour commencer que f a au plus un zéro dans le pavé $\boldsymbol{x}$ et même que $\boldsymbol{f}'(\boldsymbol{x})$ est régulière. Une méthode couramment utilisée est la méthode de Gauss-Seidel avec pré-multiplication par une matrice C (cf. §6.3). On peut montrer que si $C\boldsymbol{A}$ est une H-matrice et si $\tilde{x} = \text{mid}(\boldsymbol{x})$ alors cette itération définit une suite qui soit converge vers l'unique zéro de f dans $\boldsymbol{x} = \boldsymbol{x}^0$, soit devient la suite stationnaire $\boldsymbol{x}^k = \emptyset$ à partir d'un certain rang. Une telle suite est dite *fortement convergente*.

5.2.2 Cas où la fonction admet plusieurs zéros

Quand la fonction f admet plusieurs zéros dans le pavé $\boldsymbol{x}$, la fonction Jacobienne cesse d'être inversible sur $\boldsymbol{x}$ et l'on ne pourra plus appliquer les théorèmes de convergence précédents. La technique dite de bissection utilisée dans ce cas consiste à découper le pavé en sous-pavés $\boldsymbol{x}_i$ qui soit peuvent être éliminés immédiatement si $\boldsymbol{f}(\boldsymbol{x}_i)$ ne contient pas 0, soit peuvent se voir appliquer avec succès une méthode de Newton par intervalles, soit enfin seront découpés à leur tour.

Cela réclame de stocker les pavés en attente de traitement dans une file d'attente. De même, le résultat de l'algorithme sera une liste de pavés satisfaisant un critère d'arrêt à préciser et susceptibles de contenir un zéro de f. Le squelette de l'algorithme est le suivant, il est plus détaillé que l'algorithme qui était présenté de façon sommaire au §5.1 pour éviter les redites.

5.2.3 Algorithme de Newton par intervalles

Données : $\boldsymbol{f}$ une extension intervalle de f et $\boldsymbol{f}'$ une extension intervalle de f'
$\boldsymbol{x}$ un pavé de recherche
Résultat : Res une liste de pavés susceptibles de contenir un zéro de f
Initialisations :
$\mathcal{L} := \{\boldsymbol{x}\}$ la liste des pavés en attente de traitement
Res $:= \emptyset$
tant que $\mathcal{L} \neq \emptyset$ **faire**
sortir un pavé $\boldsymbol{x}$ de $\mathcal{L}$
$\boldsymbol{x}' := N(\tilde{x}, \boldsymbol{x})$ avec N un pas d'une méthode de Newton
si $\boldsymbol{x}' \neq \emptyset$ alors
si $N(\tilde{x}', \boldsymbol{x}') \subset \boldsymbol{x}'$ alors " $\boldsymbol{x}'$ contient un zéro " // cf. §5.3
si $N(\tilde{x}', \boldsymbol{x}') \subset \text{int}(\boldsymbol{x}')$ alors " $\boldsymbol{x}'$ contient un unique zéro "
si $\boldsymbol{x}'$ satisfait le critère d'arrêt alors
ranger $\boldsymbol{x}'$ dans Res
sinon
si $\boldsymbol{x}'$ est assez réduit par rapport à $\boldsymbol{x}$ alors
ranger $\boldsymbol{x}'$ dans $\mathcal{L}$
sinon
$(\boldsymbol{x}_1, \boldsymbol{x}_2) := \text{split}(\boldsymbol{x}')$ (bissection de $\boldsymbol{x}'$)
si $\boldsymbol{f}(\boldsymbol{x}_1) \ni 0$ alors ranger $\boldsymbol{x}_1$ dans $\mathcal{L}$
idem pour $\boldsymbol{x}_2$

Il reste à préciser le test d'arrêt, le test de réduction et la façon de découper $\boldsymbol{x}'$ en deux ou éventuellement plus.

Le test d'arrêt doit mesurer si $\boldsymbol{x}'$ est assez petit, si $\boldsymbol{f}(\boldsymbol{x}')$ est petit également et si une bissection peut améliorer le résultat trouvé [4]. Le test sur la taille de $\boldsymbol{x}'$ mesure la précision relative obtenue pour le zéro contenu dans $\boldsymbol{x}'$, il s'écrit $\max_i w_r(\boldsymbol{x}'_i) \leq \varepsilon$ avec, pour chaque composante $\boldsymbol{x}'_i$ de $\boldsymbol{x}'$, $w_r(\boldsymbol{x}'_i) = w(\boldsymbol{x}'_i)/|\text{mid}(\boldsymbol{x}'_i)|$ si $\text{mid}(\boldsymbol{x})'_i \neq 0$ et $w_r(\boldsymbol{x}'_i) = w(\boldsymbol{x}'_i)$ sinon ; ce test remplace le test sur la stagnation des itérés utilisé pour l'algorithme de Newton flottant. Le seuil ε pour la taille de $\boldsymbol{x}'$ peut aller jusqu'à u la précision machine puisque des bissections répétées permettent d'atteindre cette valeur. Le test $w(\boldsymbol{f}(\boldsymbol{x}')) \leq \varepsilon'$ indique si le résidu $\boldsymbol{f}(\boldsymbol{x}')$ est assez proche de 0 : il s'agit d'un test sur la précision absolue du résidu. Le seuil ε' pourrait aller jusqu'à η le plus petit nombre flottant strictement positif, cependant l'accumulation d'erreurs d'arrondi et de surencadrements liés à l'arithmétique par intervalles interdisent d'atteindre cette valeur. La dernière partie du test d'arrêt détermine si une bissection peut permettre d'affiner cette valeur, selon les auteurs [4, 40], ce test compare $\boldsymbol{f}(\boldsymbol{x}')$ à $\boldsymbol{f}(\text{mid}(\boldsymbol{x}'))$ ou à $\boldsymbol{f}(\boldsymbol{x}_1)$ et $\boldsymbol{f}(\boldsymbol{x}_2)$ avec $(\boldsymbol{x}_1, \boldsymbol{x}_2) = \text{split}(\boldsymbol{x}')$: une bissection est inutile si $w(\boldsymbol{f}(\text{mid}(\boldsymbol{x}'))) \geq$

$\nu w(\boldsymbol{f}(\boldsymbol{x}'))$ ou si $\max(w(\boldsymbol{f}(\boldsymbol{x}_1)), w(\boldsymbol{f}(\boldsymbol{x}_2))) \geq w(\boldsymbol{f}(\boldsymbol{x}'))$. La première solution impose de fixer une valeur de ν convenable, la seconde de calculer éventuellement inutilement $\boldsymbol{f}(\boldsymbol{x}_1)$ et $\boldsymbol{f}(\boldsymbol{x}_2)$.

Le test de réduction mesure si un pas de la méthode de Newton par intervalles $\boldsymbol{x}' := N(\boldsymbol{x}, \tilde{x})$ a réduit le pavé $\boldsymbol{x}$ ou non. En général ce test s'écrit $w(\boldsymbol{x}') \leq \alpha w(\boldsymbol{x})$ avec α un paramètre $\in]0; 1]$ à fixer également ; dans le cas où $w(\boldsymbol{x}') > \alpha w(\boldsymbol{x})$, le pavé $\boldsymbol{x}'$ est insuffisamment réduit et il est coupé en deux avant d'être inséré dans la liste $\mathcal{L}$ des pavés en attente. Ceci permet d'assurer la terminaison de l'algorithme puisque tout pavé finira par avoir une largeur relative inférieure au seuil prescrit.

Enfin, la stratégie de bissection a fait l'objet de nombreuses études théoriques et expérimentales [16, 37]. L'idée la plus simple consiste à couper en deux le côté ayant la plus grande largeur afin d'assurer une décroissance de la largeur du pavé. On peut également découper le côté sur lequel la fonction varie le plus, c'est-à-dire le côté $\boldsymbol{x}_i$ pour lequel $w(\boldsymbol{f}'(\boldsymbol{x})_i).w(\boldsymbol{x}_i)$ est maximal, dans l'espoir d'éliminer rapidement une moitié. Enfin, dans le cas où des divisions par des intervalles contenant 0 ont créé deux sous-intervalles dans l'une des dimensions du pavé, la bissection peut se résumer à retourner les deux sous-pavés correspondant. Il faut cependant que le " trou" entre les deux sous-intervalles soit assez large pour être intéressant et également assez centré pour que les deux sous-pavés soient assez réduits.

5.3 Propriétés de la méthode de Newton par intervalles : preuve de l'existence et de l'unicité d'un zéro

Notons $N(\boldsymbol{x}, \tilde{x})$ le pavé $\tilde{x} - \Sigma_{\exists\exists}(\boldsymbol{f}'(\boldsymbol{x}), f(\tilde{x}))$. L'existence, l'existence et unicité ou l'absence de zéro de f dans $\boldsymbol{x}$ sont données par le théorème suivant.
Théorème (Moore et Nickel).
Soit $f : \mathbb{R}^n \to \mathbb{R}^n$ et soit $\boldsymbol{f}$ une extension intervalle de f (définie au §4.1), si $\boldsymbol{f}$ est lipschitzienne (au sens intervalle) et si $\boldsymbol{f}'(\boldsymbol{x})$ ne contient pas 0, alors :
– tout zéro de f dans $\boldsymbol{x}$ appartient aussi à $N(\boldsymbol{x}, \tilde{x})$ pour $\tilde{x} \in \boldsymbol{x}$;
– si $N(\boldsymbol{x}, \tilde{x}) \cap \boldsymbol{x} = \emptyset$ alors f n'admet pas de zéro dans $\boldsymbol{x}$;
– soit $\tilde{x} \in \text{int}(\boldsymbol{x})$ l'intérieur de $\boldsymbol{x}$, si $N(\boldsymbol{x}, \tilde{x}) \subset \boldsymbol{x}$ alors f admet au moins un zéro dans $\boldsymbol{x}$;
– soit $\tilde{x} \in \text{int}(\boldsymbol{x})$, si $N(\boldsymbol{x}, \tilde{x}) \subset \text{int}(\boldsymbol{x})$ alors f admet un seul zéro dans $\boldsymbol{x}$.
Les deux premières propriétés sont des propriétés de garantie des résultats et la troisième est une formulation par intervalle du théorème de Brouwer/Schauder. Il est remarquable que les hypothèses de ce théorème soient faciles à vérifier en arithmétique par intervalles : comparer $\boldsymbol{x}$ et un surencadrement de $N(\tilde{x}, \boldsymbol{x})$ est typiquement ce que cette arithmétique rend possible. La quatrième propriété découle

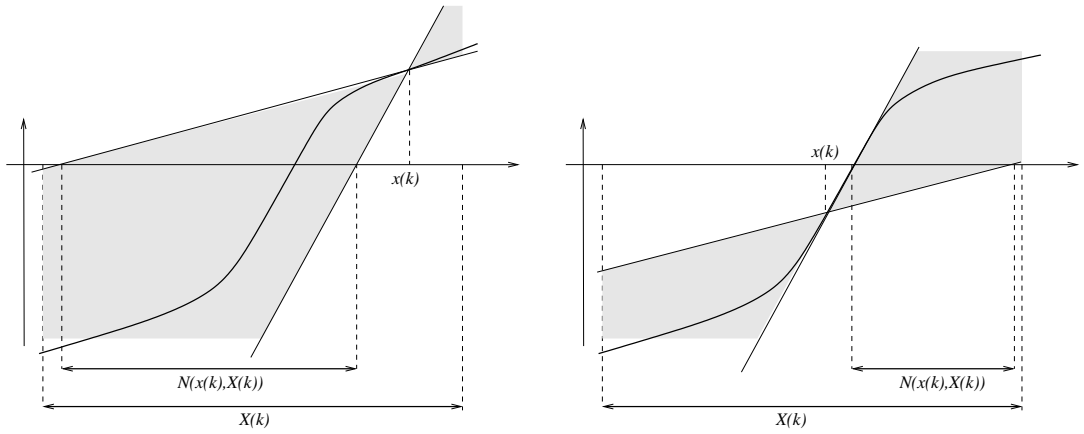


FIG. 11 – Influence du " gonflage " sur l'applicabilité du théorème de Moore et Nickel : à gauche, le point $\tilde{x}$ est au bord de l'intervalle et le meilleur $N(\tilde{x}, \boldsymbol{x})$ est presque égal à $\boldsymbol{x}$, à droite le point $\tilde{x}$ est plus loin des bords après " gonflage " de $\boldsymbol{x}$ et le meilleur $N(\tilde{x}, \boldsymbol{x})$ est plus réduit, ce qui correspond à de meilleures chances que $N(\tilde{x}, \boldsymbol{x})$ soit inclus dans l'intérieur de $\boldsymbol{x}_\varepsilon$ en pratique.

du théorème du point fixe contractant.

Avec une implantation sur ordinateur, il faut bien sûr calculer $f(\tilde{x})$ par intervalles et il peut être impossible de vérifier si $N(\tilde{x}, \boldsymbol{x})$ est inclus dans l'intérieur de $\boldsymbol{x}$, ne serait-ce qu'à cause de la précision machine limitée et des arrondis vers l'extérieur qui renforcent le surencadrement. Une technique fréquemment mise en œuvre depuis son introduction par Rump [47] pour la résolution de systèmes linéaires consiste à "gonfler" l'intervalle $\boldsymbol{x}$ en $\boldsymbol{x}_\varepsilon$, avec ε un paramètre caractérisant l'augmentation de largeur, et à tester l'inclusion de $N(\tilde{x}, \boldsymbol{x}_\varepsilon)$ dans $\boldsymbol{x}_\varepsilon$. Différents choix pour $\boldsymbol{x}_\varepsilon$ sont par exemple $\boldsymbol{x}_\varepsilon = \boldsymbol{x} + w(\boldsymbol{x})[-\varepsilon;\varepsilon] + [-\eta;\eta]$ avec $\varepsilon = u$ la précision machine et η le plus petit nombre flottant strictement positif, ou bien $\boldsymbol{x}_\varepsilon = \boldsymbol{x} + w(\boldsymbol{x})[-\varepsilon;\varepsilon]$ avec $\varepsilon = 0,25$ pour que le "gonflage" soit plus net, cf. [28]. Cela permet de recentrer $\tilde{x}$ et de réduire $N(\tilde{x}, \boldsymbol{x})$, comme indiqué sur la figure 11.

5.4 Cas de plusieurs zéros proches ou d'un zéro multiple

Dans le cas d'un zéro multiple, $\boldsymbol{f}'(\boldsymbol{x})$ n'est pas régulière et seule la bissection permet de progresser dans l'algorithme, ce qui signifie que la complexité dans le pire cas est exponentielle en la dimension du problème et en les seuils. De plus, les comportements connus dans le cas de l'algorithme de Newton flottant se retrouvent avec la méthode de Newton par intervalles ; en particulier, la précision atteinte sur la racine est $u^{1/m}$ si m est la multiplicité de la racine et u la précision

machine : les bissections successives de $\boldsymbol{x}$ ne permettent pas d'éliminer une partie de $\boldsymbol{x}$ mais uniquement de satisfaire le critère d'arrêt sur la largeur de $\boldsymbol{x}$. En revanche, dans le cas de racines proches, la bissection conjuguée à une évaluation de f d'ordre élevé (avec un développement de Taylor à l'ordre 1 ou 2 — cf. [2] sur l'importance d'une bonne évaluation de la fonction dans la méthode de Newton par intervalles) permet de distinguer ces racines.

6 Résolution d'un système linéaire par intervalles

Cette partie doit beaucoup à l'ouvrage de Neumaier [33] pour les algorithmes et leur analyse. Il ne sera donc pas cité systématiquement par la suite.

6.1 Définition du problème et résultats de complexité

Résoudre un système linéaire signifie, en arithmétique usuelle, se donner une matrice A de taille $n \times n$ et un vecteur $b \in \mathbb{R}^n$ et déterminer un vecteur $x \in \mathbb{R}^n$ tel que $Ax = b$. Si on se donne une matrice intervalle de taille $n \times n$ $\boldsymbol{A}$ et un vecteur intervalle $\boldsymbol{b} \in \mathbb{IR}^n$, il n'existe pas nécessairement de vecteur $\boldsymbol{x} \in \mathbb{IR}^n$ vérifiant[13] $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.
Une définition classique du problème de la résolution de systèmes linéaires par intervalles consiste à déterminer l'ensemble des vecteurs $x \in \mathbb{R}^n$ qui sont solution d'un système linéaire ponctuel $Ax = b$ avec $A \in \boldsymbol{A}$ et $b \in \boldsymbol{b}$, ou plus précisément un pavé contenant cet ensemble qui n'est pas nécessairement convexe, comme le montre l'exemple suivant [33] : l'ensemble des $x \in \mathbb{R}^2$ pour lesquels :

$$\exists A \in \boldsymbol{A} = \begin{pmatrix} [2;4] & [-1;1] \\ [-1;1] & [2;4] \end{pmatrix} \text{ et } \exists b \in \boldsymbol{b} = \begin{pmatrix} [-3;3] \\ 0 \end{pmatrix} \text{ tels que } Ax = b$$

est représenté sur la figure 12.
Dans la suite, en notant Conv E l'enveloppe convexe d'un ensemble E, nous chercherons à déterminer un pavé contenant

$$\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b}) = \text{Conv}\{x \in \mathbb{R}^n \mid \exists A \in \boldsymbol{A}, \exists b \in \boldsymbol{b}, Ax = b\}.$$

Pour mémoire, il a été mentionné au §3.5 que le problème de déterminer exactement $\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})$ est NP-dur et le déterminer à un polynôme en n près l'est également.
Autrement dit, tout comme pour le problème de l'évaluation de l'image d'une fonction sur un intervalle, l'objectif est de déterminer un surencadrement " pas trop large " de $\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})$.

[13] Si par exemple $\boldsymbol{A} = [-1;1]$ et $\boldsymbol{b} = [2;3]$, le produit de $\boldsymbol{A}$ par tout intervalle $\boldsymbol{x}$ contiendra 0 puisque $0 \in \boldsymbol{A}$ et ne pourra donc pas être égal à $\boldsymbol{b}$.

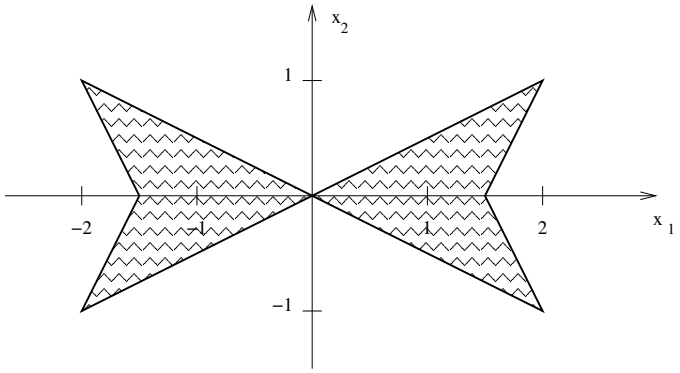


FIG. 12 – Solution d'un système linéaire intervalle.

6.2 Élimination de Gauss par intervalles

L'algorithme le plus connu pour la résolution de systèmes linéaires est probablement l'algorithme d'élimination de Gauss. On peut tenter de l'appliquer à un système linéaire intervalle $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, aussi longtemps qu'aucun pivot ne contient 0. Dans l'hypothèse où l'élimination a été poursuivie jusqu'à son terme, on obtient à la place de $\boldsymbol{A}$ une matrice triangulaire supérieure $\boldsymbol{U}$ et on peut construire une matrice triangulaire inférieure $\boldsymbol{L}$ contenant les coefficients des combinaisons linéaires de lignes effectuées pendant l'élimination. Il est à noter que l'on a simplement l'inclusion $\boldsymbol{A} \subset \boldsymbol{L}\boldsymbol{U}$ puisque $\boldsymbol{L}\boldsymbol{U} = \{L \times U \mid L \in \boldsymbol{L}, U \in \boldsymbol{U}\}$ contient des produits $L \times U$ où L et U ne proviennent pas de la factorisation d'une même matrice $A \in \boldsymbol{A}$. On peut alors trouver un surencadrement de $\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})$ en " résolvant " par descente triangulaire le système linéaire $\boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$ puis par remontée $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$.
Appliquer une stratégie de pivot partiel ne permet pas nécessairement de stabiliser l'algorithme de Gauss, comme l'illustre l'exemple suivant :

$$\boldsymbol{A} = \begin{pmatrix} [0,1;0,9] & 0 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

$\boldsymbol{A}$ admet la factorisation LU suivante, obtenue sans pivotage :

$$\boldsymbol{L} = \begin{pmatrix} 1 & 0 & 0 \\ [-10;-1/0,9] & 1 & 0 \\ 0 & -1/3 & 1 \end{pmatrix}, \quad \boldsymbol{U} = \begin{pmatrix} [0,1;0,9] & 0 & 0 \\ 0 & 3 & -1 \\ 0 & 0 & 5/3 \end{pmatrix},$$

alors que la stratégie de pivot partiel conduit à échanger les deux premières lignes

et après la première étape d'élimination on obtient :

$$\begin{pmatrix} -1 & 3 & -1 \\ 0 & [0,3;2,7] & [-0,9;-0,1] \\ 0 & -1 & 2 \end{pmatrix} \ni \begin{pmatrix} -1 & 3 & -1 \\ 0 & 0,4 & -0,8 \\ 0 & -1 & 2 \end{pmatrix}$$

où le bloc 2×2 en bas à droite est non inversible.
Comme en algorithmique numérique classique, il existe peu de résultats positifs caractérisant les matrices pour lesquelles l'élimination de Gauss peut être conduite à son terme, ils concernent essentiellement des sous-classes de matrices telles que les M-matrices qui interviennent souvent lors de la discrétisation d'EDP par exemple, ou les H-matrices, pour lesquelles on s'affranchit des contraintes de signe définissant les M-matrices (voir [33] pour les définitions et les preuves).
L'algorithme d'élimination de Gauss par intervalles peut conduire à des surencadrements du résultat beaucoup trop larges, en particulier quand les éléments de la matrice sont des intervalles larges. On a même vu au §3.5 que la surestimation peut être arbitrairement grande. Illustrons ce phénomène par un exemple :

$$\text{soient } \boldsymbol{A} = \begin{pmatrix} 1 & & 0 \\ \vdots & \ddots & \\ 1 & \dots & 1 \end{pmatrix} \text{ et } \boldsymbol{b} = \begin{pmatrix} [-\alpha;\alpha] \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

l'algorithme de descente triangulaire conduit à $\boldsymbol{x} = ([-\alpha;\alpha], [-\alpha;\alpha], [-2\alpha;2\alpha], \dots [-2^{n-2}\alpha; 2^{n-2}\alpha])^t$ alors que $\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b}) = ([-\alpha;\alpha], [-\alpha;\alpha], 0, \dots 0)^t$. On a dans ce cas une surestimation exponentielle du résultat.
Cet algorithme est revenu en usage après qu'un pré-traitement[14] lui a été adjoint. L'idée d'un pré-traitement consiste à rapprocher la matrice du système linéaire à résoudre de la matrice identité. On peut songer *a priori* à pré-multiplier un système linéaire $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ à droite et à gauche par des matrices scalaires C et C' et à résoudre $C\boldsymbol{A}C'\boldsymbol{z} = C\boldsymbol{b}$, la solution étant $\boldsymbol{x} = C'\boldsymbol{z}$; le choix $C = \text{mid}(\boldsymbol{A})^{-1}$, $C' = I$ est le pré-traitement le plus utilisé en pratique[15].
Dans le cas où $\text{mid}(\boldsymbol{A})^{-1} \times \boldsymbol{A}$ est régulière, l'algorithme d'élimination de Gauss peut lui être appliqué. Si de plus $\text{mid}(\boldsymbol{A})^{-1} \times \boldsymbol{A}$ est à diagonale strictement dominante, alors la solution obtenue par élimination de Gauss et remontée triangulaire est une approximation quadratique de $\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})$. Cependant, Mayer et Rohn [29] ont montré que dans le cas où la matrice milieu de cette matrice est

[14] Ce pré-traitement est souvent et improprement appelé " préconditionnement ".

[15] Comme en général les problèmes abordés en arithmétique par intervalles sont de petite taille, comparée à celle des problèmes résolus en utilisant l'arithmétique flottante usuelle, l'inversion d'une matrice ponctuelle par une procédure numérique n'est pas déraisonnable.

diagonale, alors un autre algorithme direct, l'algorithme de Hansen-Bliek-Rohn-Ning-Kearfott [34], donne exactement $\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})$.
Pour résumer, l'algorithme d'élimination de Gauss était connu pour donner des surestimations très larges des résultats, avant que l'on ne s'aperçoive expérimentalement qu'il se comportait bien sur des systèmes pré-traités puis qu'il soit prouvé optimal pour les M-matrices.

6.3 Méthode itérative de Gauss-Seidel par intervalles

Une autre façon d'aborder le problème, qui peut être comparée à une technique de propagation de contraintes, consiste à " résoudre " la i-ème ligne du système linéaire $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ en la i-ème variable $\boldsymbol{x}_i$: si un pavé de recherche est donné, la i-ème contrainte peut se réécrire en :

$$\boldsymbol{x}_i' = \left(\boldsymbol{b}_i - \sum_{j \neq i} \boldsymbol{A}_{i,j} \boldsymbol{x}_j \right) / \boldsymbol{A}_{i,i} \cap \boldsymbol{x}_i.$$

On en déduit l'itération série de Gauss-Seidel où k est le numéro de l'itération :

$$\boldsymbol{x}_i^{(k+1)} = \left(\boldsymbol{b}_i - \sum_{j=1}^{i-1} \boldsymbol{A}_{i,j} \boldsymbol{x}_j^{(k+1)} - \sum_{j=i+1}^{n} \boldsymbol{A}_{i,j} \boldsymbol{x}_j^{(k)} \right) / \boldsymbol{A}_{i,i} \cap \boldsymbol{x}_i^{(k)},$$

que l'on peut aussi écrire de façon matricielle par :

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{M}^{-1} \left(\boldsymbol{b} + \boldsymbol{N}\boldsymbol{x}^{(k)} \right) \cap \boldsymbol{x}^{(k)}$$

avec $\boldsymbol{M}$ la partie triangulaire inférieure de $\boldsymbol{A}$, diagonale incluse, et $\boldsymbol{N}$ l'opposé de la partie triangulaire supérieure de $\boldsymbol{A}$, diagonale exclue (on a $\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$).
Si cette itération converge, sa limite est le point fixe de l'équation :

$$\boldsymbol{x} = \boldsymbol{M}^{-1} \left(\boldsymbol{b} + \boldsymbol{N}\boldsymbol{x} \right);$$

de plus, si le pavé initial $\boldsymbol{x}^{(0)} \supset \Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})$, alors tous les itérés contiennent $\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})$ et la suite $(\boldsymbol{x}^{(k)})_k$ ainsi définie est monotone décroissante pour l'inclusion. La convergence est assurée si cette itération est contractante (cf. le théorème du point fixe), ce qui s'écrit dans ce cas $\rho(|\boldsymbol{M}|^{-1} \times |\boldsymbol{N}|) < 1$, avec ρ désignant le rayon spectral, et on a alors existence et unicité de la solution. En général cette condition est difficile à vérifier, mais, avec l'arithmétique par intervalles, le caractère contractant d'une itération peut facilement être vérifié en pratique, en testant l'inclusion d'un itéré dans le précédent.
Ici aussi, un pré-traitement peut permettre d'améliorer le comportement de l'algorithme, le plus souvent il consiste à multiplier le système à gauche par $\mathrm{mid}(\boldsymbol{A})^{-1}$.

L'algorithme itératif de Gauss-Seidel ainsi pré-traité porte le nom d'algorithme de Rump [48] ou de Hansen-Sengupta. Pour une étude des pré-multiplieurs pour l'algorithme de Gauss-Seidel, voir [3].
Dans le cas où la matrice du système éventuellement pré-traité $\boldsymbol{A}$ est une H-matrice et où mid($\boldsymbol{A}$) est diagonale, alors l'algorithme d'élimination de Gauss fournit un meilleur surencadrement de $\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})$. Une utilisation pratique de Gauss-Seidel peut alors se limiter à quelques itérations pour améliorer le résultat de l'algorithme d'élimination de Gauss ou de Hansen-Bliek-Rohn-Ning-Kearfott.
Les méthodes flottantes de type Krylov [50] ne semblent pas avoir donné lieu à des méthodes par intervalles ; l'explication en est que ces méthodes construisent des bases (bi-)orthogonales des sous-espaces de Krylov et que la notion d'orthogonalité a peu de sens en arithmétique par intervalles.

7 Résolution de contraintes

La résolution de systèmes linéaires et non linéaires a également été étudiée par des spécialistes de la programmation logique sous contraintes dans le cas discret. Ils ont adapté les techniques et les notions utilisées en programmation logique au cas du calcul sur les intervalles [54, 8, 11, 20].

7.1 Algorithme de propagation - rétropropagation

Une technique couramment utilisée en programmation logique pour résoudre un ensemble de contraintesest appelée *propagation de contraintes* et son adaptation à la résolution de systèmes de contraintes réelles est présentée ci-dessous. On suppose qu'il s'agit d'une contrainte donnée par un programme et dont la valeur de sortie est fixée. Ce programme est décomposé en une suite d'instructions élémentaires de la forme $x_k := x_i \diamond x_j$ ou $x_k := \varphi(x_i)$ où les opérations réciproques de $\diamond$ ($\diamond_g^{-1}$ et $\diamond_d^{-1}$ pour les réciproques à gauche et à droite) et de φ (à savoir φ^{-1}) sont connues. Par exemple si $\diamond$ est l'addition alors $\diamond^{-1}$ est la soustraction et si $\varphi = \sin$ alors $\varphi^{-1} = \arcsin$. Le programme de résolution s'écrit :

Initialisations : initialiser les valeurs de sortie aux valeurs désirées
tant que une amélioration a été apportée **faire**
    forward propagation
        évaluer dans l'ordre les instructions élémentaires :
        $x_k := (x_i \diamond x_j) \cap x_k$ ou $x_k := \varphi(x_i) \cap x_k$
    backward propagation
        pour chaque instruction élémentaire dans l'ordre inverse
            si cette instruction élémentaire est $x_k := x_i \diamond x_j$ alors

évaluer $x_i := (x_k \diamond_d^{-1} x_j) \cap x_i$ et $x_j := (x_i \diamond_g^{-1} x_k) \cap x_j$
sinon (elle est de la forme $x_k := \varphi(x_i)$)
évaluer $x_i := \varphi{-}1(x_k) \cap x_i$

On s'arrête quand aucune variable intervalle (donnée ou intermédiaire) n'est modifiée. Cette technique permet de réduire efficacement les pavés de recherche initiaux et est utilisée notamment dans Numerica [55], Prolog IV [7] et Alias [12].

Cet algorithme est un exemple de contracteur [20] : ce sont les algorithmes employés en programmation logique sous contraintes. L'algorithme de Gauss-Seidel présenté au §6.3 peut aussi être vu comme un contracteur. Cet algorithme n'utilise que la phase de rétropropagation et ne met à jour que la variable $\boldsymbol{x}_i$ à l'aide de la i-ème contrainte. Les contracteurs ou algorithmes contractants réduisent l'ensemble initial sans jamais procéder à des bissections et ils ont des complexités polynomiales. Cependant, ils s'arrêtent quand ils ne parviennent plus à réduire l'ensemble courant et non quand ils sont parvenus à l'optimum. Un bon contracteur est donc un algorithme qui s'arrête avec un ensemble petit au sens de l'inclusion. Les notions présentées au §7.2 permettent de comparer les ensembles obtenus et par conséquent les contracteurs.

Si jamais on désire réduire encore le résultat obtenu par un contracteur, on peut sacrifier la complexité polynomiale et procéder à une bissection. On applique ensuite le contracteur aux deux sous-pavés pour les réduire à nouveau etc.

La méthode de Newton tout comme la méthode de propagation des contraintes, et les contracteurs de façon générale, sont, comme leur nom l'indique, des méthodes contractantes, qui s'appliquent donc particulièrement bien à l'arithmétique par intervalles. De plus, les théorèmes de point fixe (théorème de Brouwer, théorème de point fixe contractant) deviennent des théorèmes effectifs avec cette arithmétique et ces algorithmes, puisqu'il est facile de vérifier une inclusion telle que $\boldsymbol{f}(\boldsymbol{x}) \subset \boldsymbol{x}$ ou $\boldsymbol{f}(\boldsymbol{x}) \subset \text{int}(\boldsymbol{x})$ pour pouvoir en conclure à l'existence et éventuellement à l'unicité du résultat cherché.

7.2 Notions adaptées de la programmation logique sous contraintes

Outre les techniques utilisées en programmation logique sous contraintes, quelques notions ont été importées. En particulier les notions de consistance ont été adaptées. Elles permettent de classifier les méthodes employées et de les comparer, ou, comme montré dans l'exemple ci-dessous, de comparer la qualité des intervalles

résultats. [54, 8, 11].

Par exemple, une contrainte de la forme $c(x_1, \ldots, x_n)$ est dite consistante par arc avec l'ensemble E, qui est supposé être le résultat calculé, si et seulement si l'ensemble E est le plus petit ensemble de n-uplets satisfaisant la contrainte c ; plus formellement, cela se traduit par le fait que pour tout $x \in E$ et pour tout $i \in \{1, \ldots, n\}$, il existe x_j avec $j \neq i$ tels que $(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) \in E$ et $c(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ est satisfaite. L'adaptation de cette notion au cas des intervalles a donné naissance aux notions de consistance par enveloppe convexe ou *hull consistency*, encore appelée consistance 2B, ou de consistance par pavé ou *box consistency*.

Une contrainte de la forme $c(x_1, \ldots, x_n) = 0$ avec $x_i \in I\!R$ pour $1 \leq i \leq n$ est dite consistante par pavé avec le pavé $\boldsymbol{B} = \boldsymbol{B}_1 \times \ldots \boldsymbol{B}_n \in I\!I\!R^n$, $\boldsymbol{B}_i \in I\!I\!R$ pour $1 \leq i \leq n$ si et seulement si :

$$\forall x_i \in \boldsymbol{B}_i,\ \exists x_j \in \boldsymbol{B}_j \text{ pour } j \neq i \ / \ c(x_1, \ldots x_{i-1}, x_i, x_{i+1}, \ldots x_n) = 0.$$

Autrement dit le pavé $\boldsymbol{B}$ est l'enveloppe convexe fermée de l'ensemble pour lequel la contrainte est consistante par arc.

8 Conclusion

8.1 Autres problèmes bien résolus en arithmétique par intervalles

Le choix des problèmes et les algorithmes de résolution présentés dans ce qui précède est motivé d'une part par la quantité de publications sur ces sujets et d'autre part par les centres d'intérêt de l'auteur et est donc relativement subjectif. On peut mentionner d'autres problèmes traités avec succès par l'arithmétique par intervalles, comme la recherche des valeurs et vecteurs propres d'une matrice symétrique, l'inversion ensembliste ou l'intégration d'équations différentielles ordinaires avec conditions initiales.

Valeurs et vecteurs propres d'une matrice symétrique

Beaumont [6] détermine une caractérisation des éléments propres d'une matrice symétrique à l'aide d'un ensemble de programmes linéaires, en utilisant des majorations pour remplacer les termes quadratiques par des termes linéaires, et il calcule ensuite une boîte oblique contenant ces éléments propres.

Inversion ensembliste

Le problème de l'inversion ensembliste consiste, pour une fonction donnée $f : I\!R^m \rightarrow I\!R^n$ et un ensemble $Y \subset I\!R^n$, à déterminer des encadrements par l'intérieur

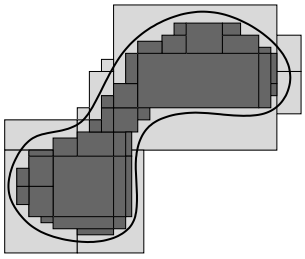


FIG. 13 – Illustration des encadrements intérieur et extérieur.

$\underline{X}$ et par l'extérieur $\overline{X}$ de l'ensemble X tel que $f(X) = Y$. La méthode SIVIA proposée par Jaulin [20] et illustrée figure 13 détermine $\underline{X}$ et $\overline{X}$ en utilisant l'arithmétique par intervalles : $\underline{X}$ est l'union de pavés de $\mathbb{R}^m$ tels que $\boldsymbol{f}(\underline{X}) \subset Y$ avec $\boldsymbol{f}$ une extension intervalle de f. L'encadrement extérieur $\overline{X}$ est l'union des pavés constituant $\underline{X}$ et des pavés dont l'image par $\boldsymbol{f}$ rencontre Y.

Intégration des équations différentielles ordinaires

L'une des premières applications de l'arithmétique par intervalles a été l'intégration des équations différentielles ordinaires avec conditions initiales. Le problème de Cauchy est la recherche d'une solution x satisfaisant :

$$\left\{ \begin{array}{lcl} x' & = & f(t,x), \\ x(t_0) & = & x_0. \end{array} \right.$$

On prouve l'existence et l'unicité de la solution (sous certaines conditions) en montrant que l'opérateur de Picard :

$$P \ : \ \varphi \in \mathcal{C}^1 \mapsto P(\varphi) : t \mapsto x_0 + \int_{t_0}^{t} f(u, \varphi(u)) du$$

est contractant. On peut donc itérer cet opérateur pour construire, en arithmétique par intervalles, la solution au problème de Cauchy [32].

Cette approche semble plus adaptée à l'arithmétique par intervalles que l'utilisation des méthodes de type Euler ou Runge-Kutta qui ont une fâcheuse tendance à donner des encadrements de plus en plus larges de $x(t)$ au fur et à mesure que t s'éloigne du point de départ t_0.

8.2 Conclusion et credo personnel

L'arithmétique par intervalles constitue une bonne approche pour répondre à l'exigence de fiabilité des calculs. En effet, elle repose sur le principe que tout calcul retourne un encadrement garanti de son résultat. De plus, elle peut être implantée

efficacement sur ordinateur et une panoplie d'algorithmes numériques ont été développés spécifiquement pour tirer parti de cette arithmétique. Il serait naïf de croire que les algorithmes utilisés en arithmétique flottante donnent des résultats probants sur des intervalles, le plus souvent les encadrements obtenus sont trop pessimistes. En revanche, des algorithmes conçus pour l'arithmétique par intervalles, le plus souvent des algorithmes itératifs basés sur une itération contractante, fournissent des informations et des résultats inaccessibles en flottant : rappelons le problème de l'optimisation globale d'une fonction continue pour lequel les algorithmes flottants retournent un optimum local. On peut également mentionner le problème de la résolution de systèmes non linéaires traité dans cet article : en utilisant l'arithmétique par intervalles, on peut non seulement déterminer toutes les solutions mais également prouver leur existence ou leur unicité, en utilisant les théorèmes de point fixe qui deviennent effectifs avec une telle arithmétique.

Toute médaille a son revers, celui de l'arithmétique par intervalles est de retourner des encadrements parfois trop larges des résultats. Nous croyons beaucoup aux possibilités offertes par la combinaison de l'arithmétique par intervalles et de l'arithmétique multi-précision, qui allie fiabilité et précision. En effet, dès lors que l'on dispose d'une itération contractante, seule la précision du calcul (en faisant fi de la complexité exponentielle du calcul) constitue un obstacle à l'obtention de résultats aussi précis que l'on veut. Nos premières expériences semblent montrer que le surcoût dû à la précision multiple est compensé par l'économie de certains calculs inutiles.

Un dernier aspect qu'il est important de développer, ne serait-ce que pour convaincre par l'exemple de futurs utilisateurs, est la résolution de problèmes pratiques. En France, on peut citer l'intérêt croissant des automaticiens pour le calcul ensembliste en général et le calcul par intervalles en particulier ; le livre de Jaulin *et al.* [20] en fait foi, tout comme les travaux du groupe de travail *Méthodes ensemblistes pour l'automatique*, cf. `http://www-lag.ensieg.inpg.fr/gt-ensembliste/`.

Pour en savoir plus : la communauté d'arithmétique par intervalles a son site Web, assez complet : `http://www.cs.utep.edu/interval-comp/`.

Références

[1] R. Baker Kearfott. *Rigorous global search : continuous problems*. Kluwer, 1996.

[2] R. Baker Kearfott. Empirical evaluation of innovations in interval branch and bound algorithms for nonlinear systems. *SIAM J. Sci. Comput.*, 18(2) :574–594, 1997.

[3] R. Baker Kearfott, C. Hu, and M. Novoa. A review of preconditioners for the interval Gauss-Seidel method. *Interval Computations*, 1(1) :59–85, 1991.

[4] R. Baker Kearfott and G.W. Walster. On stopping criteria in verified nonlinear systems or optimization algorithms. *ACM TOMS*, 26(3) :373–389, 2000.

[5] E. Baumann. Optimal centered forms. *BIT*, pages 80–87, 1988.

[6] O. Beaumont. *Algorithmique pour les intervalles : comment obtenir un résultat sûr quand les données sont incertaines*. PhD thesis, IRISA, Université de Rennes 1, France, 1999.

[7] F. Benhamou, P. Bouvier, A. Colmerauer, H. Garetta, B. Giletta, J.-L. Massat, G.A. Narboni, S. N'Dong, R. Pasero, J.-F. Pique, Touraïvane, M. Van Caneghem, E. Vétillard, and J. Zhou. Manuel de Prolog IV. Technical report, PrologIA, 1996.

[8] F. Benhamou, F. Goualard, and L. Granvilliers. Revising hull and box consistency. In *International Conference on Logic Programming*, pages 230–244. The MIT Press, 1999.

[9] CANT Research Group. Arithmos : a reliable integrated computational environment (still under development). University of Antwerpen, Belgium, `http://win-www.uia.ac.be/u/cant/arithmos`, 2001.

[10] O. Caprani and K. Madsen. Mean value forms in interval analysis. *Computing*, 25 :147–154, 1980.

[11] H. Collavizza, F. Delobel, and M. Rueher. Comparing partial consistencies. *Reliable Computing*, 5(1) :1–16, 1999.

[12] Inria Coprin project. ALIAS (Algorithms Library of Interval Analysis for Systems). `http://www-sop.inria.fr/coprin/developpements/main.html`, 2003.

[13] A.A. Gaganov. Computational complexity of the range of the polynomial in several variables. *Cybernetics*, pages 418–425, 1985.

[14] A. Griewank. *Evaluating Derivatives - Principles and Techniques of Algorithmic Differentiation*. SIAM, 2000.

[15] M. Grimmer, K. Petras, and N. Revol. Multiple precision interval packages : Comparing different approaches. In *Lecture Notes in Computer Science*, volume 2991, pages 64–90, 2004.

[16] E. Hansen. *Global optimization using interval analysis*. Marcel Dekker, 1992.

[17] B. Hayes. A Lucid Interval. *Scientific American*, 91(6) :484–488, 2003.

[18] P. Hertling. A lower bound for range enclosure in interval arithmetic. In *Real Numbers and Computers 3*, 1998.

[19] IBM. *IBM High-Accuracy Arithmetic Subroutine Library (ACRITH)*, 1986.

[20] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied interval analysis*. Springer Verlag, 2001.

[21] W. Kahan. A more complete interval arithmetic. Lecture notes for a summer course at the University of Michigan, 1968.

[22] J. Keiper. Interval arithmetic in Mathematica. *Interval Computations*, (3), 1993.

[23] R. Klatte, U. Kulisch, C. Lawo, M. Rauch, and A. Wiethoff. *C-XSC a C++ class library for extended scientific computing*. Springer Verlag, 1993.

[24] O. Knueppel. PROFIL - Programmer's Runtime Optimized Fast Interval Library. Technical Report Bericht 93.4, Technische Universität Hamburg-Harburg, 1993.

[25] O. Knueppel. PROFIL/BIAS - a fast interval library. *Computing*, 53(3-4) :277–287, 1994.

[26] V. Lefèvre, P. Pelissier, and P. Zimmermann. The MPFR library (v. 2.0.3). `http://www.mpfr.org`, 2004.

[27] M. Lerch, G. Tischler, J. Wolff von Gudenberg, W. Hofschuster, and W. Krämer. *The interval library filib++ 2.0*. Universität Wuppertal, Germany, 2001.

[28] G. Mayer. Epsilon-inflation in verification algorithms. *Journal of Computational and Applied Mathematics*, 60 :147–169, 1995.

[29] G. Mayer and J. Rohn. Feasibility of the preconditioned interval Gaussian algorithm. In *SCAN,Budapest, Hungary*, page 105, 1998.

[30] R. Moore. *Interval arithmetic and automatic error analysis in digital computing*. PhD thesis, Applied Math Statistics Lab., Report 25, Stanford, 1962.

[31] R.E. Moore. *Interval analysis*. Prentice Hall, 1966.

[32] R.E. Moore. *Methods and applications of interval analysis*. SIAM Studies in Applied Mathematics, 1979.

[33] A. Neumaier. *Interval methods for systems of equations*. Cambridge University Press, 1990.

[34] A. Neumaier. A simple derivation of the Hansen-Bliek-Rohn-Ning-Kearfott enclosure for linear interval equations. `http://solon.cma.univie.ac.at/\~{}neum`, 1998.

[35] J.-C. Paoletti. Fortran Aquarels. RAP.TS.93.SEP.34, v.1-0, Simulog, 1993.

[36] H. Ratschek and J. Rokne. *New computer methods for global optimization*. Ellis Horwood Ltd, 1988.

[37] D. Ratz. Improved techniques for gap-treating and box-splitting in interval Newton Gauss-Seidel steps for global optimization with validation. *Zeitschrift für Angewandte Mathematik und Mechanik*, 76(S1) :323–326, 1996.

[38] D. Ratz. On extended interval arithmetic and inclusion isotonicity. Technical report, Institut für Angewandte Mathematik, Universität Karlsruhe, Germany, 1996.

[39] N. Revol. Arithmétique par intervalles. *Calculateurs Parallèles*, 13 :387–426, 2001.

[40] N. Revol. Interval newton iteration in multiple precision for the univariate case. *Numerical Algorithms*, 34(2) :417–426, 2003.

[41] N. Revol and F. Rouillier. The MPFI library (v. 1.3.2). `http://perso.ens-lyon.fr/nathalie.revol/software.html`, 2004.

[42] J. Rohn. Interval matrices : singularity and real eigenvalues. *SIAM J. Matrix Anal. Appl.*, 14(1) :82–91, January 1993.

[43] J. Rohn. Complexity of solving linear interval equations. Technical Report 636, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, May 1995.

[44] J. Rohn. Linear Interval Equations : Computing Sufficiently Accurate Enclosures is NP-Hard. Technical Report 621, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic, 1995.

[45] J. Rohn. Complexity of some linear problems with interval data. Technical Report 687, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, October 1996.

[46] J. Rohn and V. Kreinovich. Computing exact componentwise bounds on solutions of linear systems with interval data is NP-hard. *SIAM J. Matrix Anal. Appl.*, 16(2) :415–420, 1995.

[47] S. Rump. *Kleine Fehlerschranken bei Matrixproblemen*. PhD thesis, Universität Karlsruhe, 1980.

[48] S. Rump. *Topics in validated computations, J. Herzberger ed.*, chapter Verification methods for dense and sparse systems of equations. Elsevier, 1994.

[49] S. Rump. Fast and parallel interval arithmetic. *BIT*, 39(3) :534–554, 1999.

[50] Y. Saad. *Iterative methods for sparse linear systems*. PWS Publishing, 1996.

[51] SUN Microsystems, Inc. *C++ Interval Arithmetic Programming Reference*, 2000.

[52] SUN Microsystems, Inc. *Interval Arithmetic Programming Reference - Sun Workshop 6 Fortran 95*, 2000.

[53] T. Sunaga. Theory of interval algebra and its application to numerical analysis. *RAAG Memoirs, Ggujutsu Bunken Fukuy-kai. Tokyo*, 2 :29–46, 1958.

[54] P. Van Hentenryck, D. Mcallester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM J. Numer. Anal.*, 34(2) :797–827, 1997.

[55] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica, a modeling language for global optimization*. MIT Press, 1997.

[56] R. C. Young. The algebra of many-valued quantities. *Mathematische Annalen*, 104 :260–290, 1931.

Arithmétique des ordinateurs

Arnaud Tisserand
Arénaire INRIA LIP

École Jeunes Chercheurs en Algorithmique et Calcul Formel
Grenoble, 29 mars – 2 avril 2004

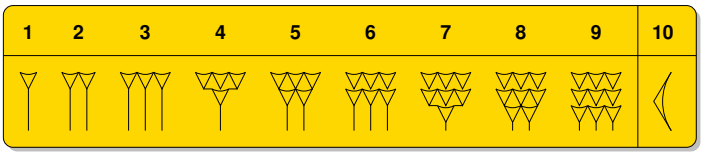
Plan de l'exposé

Partie 1

Introduction

L'arithmétique chez les Babyloniens

Utilisation d'un système de position (le premier de l'histoire) en base 60 avec les chiffres suivants (et la base auxiliaire 10) :



Exemple de codage :

$= 33 \times 60 + 24 = 2004$

Système de position en base β sur n chiffres (pour des entiers naturels) :

$$x = x_{n-1}x_{n-2}\cdots x_1x_0 = \sum_{i=0}^{n-1} x_i\,\beta^i$$

Oh, la belle table de multiplication. . .

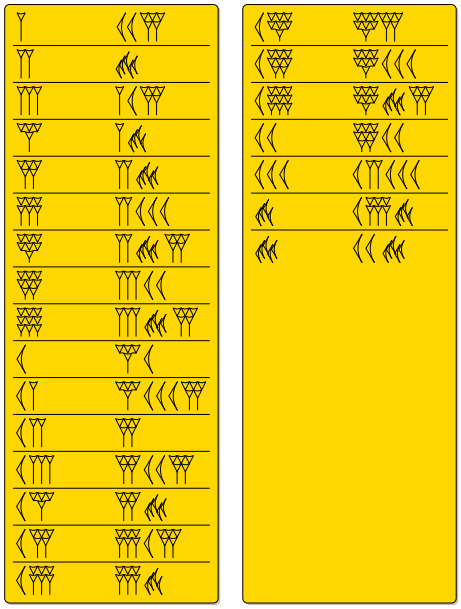
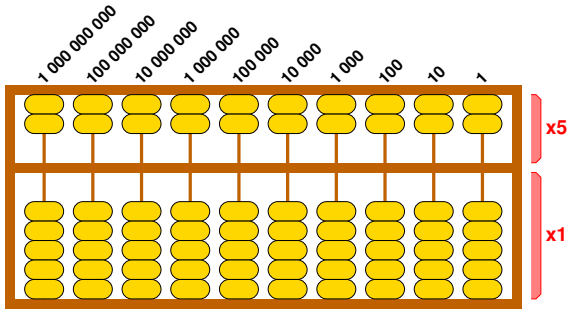


Illustration de la table de multiplication par 25 trouvée à Suse et datée du II^{e} millénaire av. J.-C (conservée au Musée du Louvre).

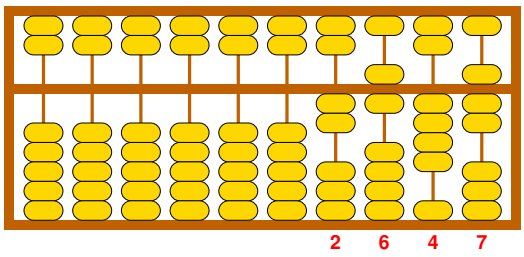
Remarque : seuls les produits par (1), 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30, 40, 50 sont nécessaires sur les 59 possibles.

Le boulier chinois : position de repos (0)



- Les colonnes représentent les unités, les dizaines, les centaines, les milliers,. . . de la droite vers la gauche
- Les boules sont activées lorsqu'elles sont au plus proche de la barre du milieu, et désactivées sinon (0 est donc représenté ci-dessus)
- Les boules situées au dessus de la barre horizontale du milieu sont multipliées par 5 (celles de dessous par 1).

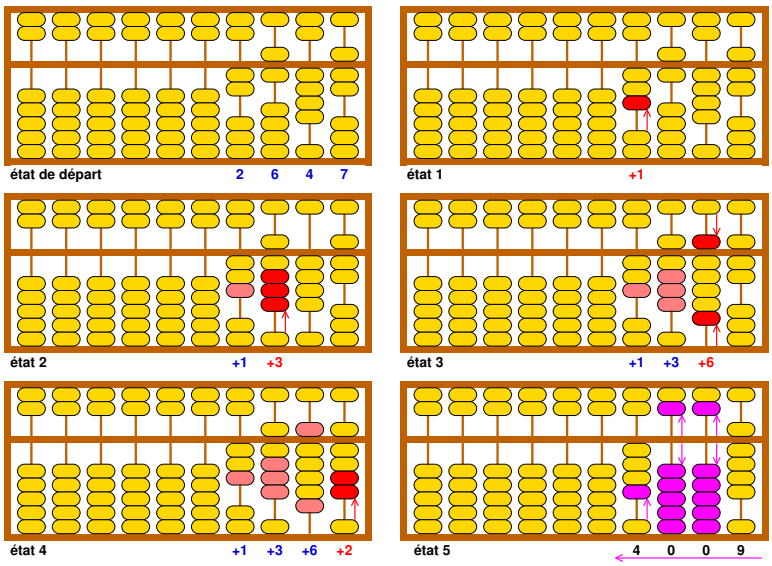
Le boulier chinois : écriture de 2647



La première illustration connue d'un boulier chinois dans un livre date de 1175. Il existe des dispositifs proches (tablettes avec des rainures sur lesquelles on déplace des petits cailloux) depuis bien plus longtemps.

Il existe des concours de rapidité de calcul sur des bouliers. Le japonais Yoshio Kogima a effectué correctement 50 divisions, avec des opérandes de 5 à 7 chiffres, en 1 min 18 s et 4 centièmes sur un boulier japonais (*soroban*) !

Le boulier chinois : 2647+1362



Arithmétique des ordinateurs

Étude et conception de "moyens" pour effectuer les calculs de base en machine.

- unités de calcul matérielles :
  - additionneur/soustracteur, multiplieur, diviseur, . . .
  - unités flottantes
  - opérateurs spécifiques (ex : filtres pour traitement du signal)
- support logiciel pour les calculs de base :
  - bibliothèques mathématiques de base (libm)
  - bibliothèques de fonctions élémentaires ($\sin, \cos, \exp, \log, \ldots$)
  - bibliothèques multi-précision
  - bibliothèques d'arithmétique d'intervalle
- validation de la qualité numérique :
  - test et/ou preuve de la précision de calculs
  - preuve du bon comportement des opérations (dépassements, . . .)

Arithmétique des ordinateurs (suite)

Les trois aspects fondamentaux de l'arithmétique des ordinateurs :

- Systèmes de représentation des nombres :
  entier, virgule fixe, virgule flottante, redondant, grande base, système logarithmique, système modulaire, corps finis. . .
- Algorithmes de calcul :
  addition–soustraction, multiplication, division, PGCD, racine carrée, fonctions élémentaires ($\sin, \cos, \exp, \log \ldots$), opérateurs composites (ex : $1/\sqrt{(x^2+y^2)}$), opérateurs spécifiques (FIR, DCT, crypto), algorithmes numériques, preuves de programmes. . .
- Maîtriser les implantations :
  cibles logicielles et matérielles, support arithmétique dans les langages de programmation, validation, test, optimisation des performances (vitesse, mémoire, surface de circuit, temps réel, consommation d'énergie). . .

Arithmétique des ordinateurs (suite)

Liens avec :

- architecture des ordinateurs
- micro-électronique
- outils CAO de circuits
- algorithmes numériques
- calcul formel
- optimisation globale
- théorie des nombres
- preuves formelles
- . . .

Arithmétique des ordinateurs (suite)

Exemples de sujets de recherche dans l'équipe/projet Arénaire :

- bibliothèque de fonctions élémentaires avec arrondi correct
- bibliothèque d'arithmétique d'intervalle en multi-précision
- bibliothèque pour le support flottant dans les processeurs entiers
- opérateurs de cryptographie pour circuits FPGA
- unités flottantes et logarithmiques pour circuits FPGA
- opérateurs arithmétiques matériels à basse consommation d'énergie
- arithmétique corps finis pour bibliothèque d'algèbre linéaire
- preuves automatiques d'opérations arithmétiques

Système logarithmique

Un nombre est représenté par son signe et le logarithme de sa valeur absolue écrit en virgule fixe (le nombre 0 doit être représenté par un codage spécial).

Les opérations dans ce système s'effectuent en utilisant :

$$\begin{aligned}
\log_2(a \times b) &= \log_2 a + \log_2 b \\
\log_2(a \div b) &= \log_2 a - \log_2 b \\
\log_2(a \pm b) &= \log_2 a \pm \log_2(1 + 2^{\log_2 b - \log_2 a}) \\
\log_2(a^q) &= q \times \log_2 a
\end{aligned}$$

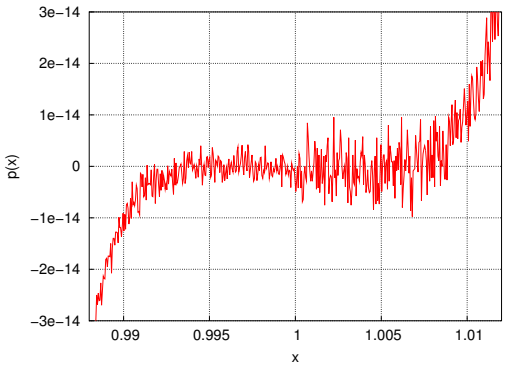
où les fonctions $\log_2(1 + 2^x)$ et $\log_2(1 - 2^x)$ sont tabulées ou approchées.

Applications en traitement du signal et en contrôle numérique. Il y avait même un projet européen pour concevoir un processeur avec des unités de calcul 32 bits en système logarithmique.

Petits problèmes numériques : un polynôme rebelle ?

Avec un petit programme C, calculons des valeurs de $p(x)$ ci-dessous pour x entre 0.988 et 1.012, et traçons la courbe correspondante.

$$p(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$$

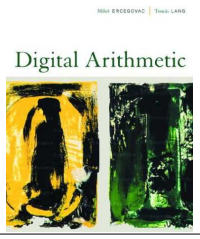


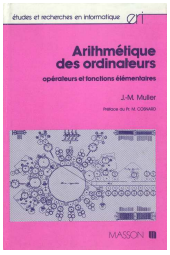
Pour en savoir plus. . .

Digital Arithmetic

Milos Ercegovac et Tomas Lang

2003
Morgan Kaufmann
ISBN : 1–55860–798–6





Arithmétique des ordinateurs

Jean–Michel Muller

1989
Masson
ISBN : 2–225–81689–1

Autres références

- Livre de G. Guitel : Histoire comparée des numérations écrites, Flammarion, 1975
- Livre de G. Ifrah : Histoire universelle des chiffres, Robert Lafond, 1994
- Document *Les Langages Numériques* de Jean Vuillemin : `http://www.di.ens.fr/~jv/HomePage/pdf/langnum.pdf`
- Numéro spécial collectif de la revue *Réseaux et systèmes répartis, calculateurs parallèles* sur l'*arithmétique des ordinateurs*, Hermes, 2001.
- Un site web sur les bouliers : `http://www.ee.ryerson.ca:8080/~elf/abacus/history.html`

Partie 2

Arithmétique flottante

Points abordés dans cette partie

- Problèmes historiques
- Représentation des nombres
- Comportement des opérations
- Propriétés des nombres flottants
- Problèmes de précision
- Exemples

Représentation flottante

Un nombre x est représenté en virgule flottante de base β par :

- son signe s_x (codage sur un bit : 0 pour x positif et 1 pour x négatif)
- son exposant e_x, un entier de k chiffres compris entre e_{min} et e_{max}
- sa mantisse m_x de $n+1$ chiffres

tels que

$$x = (-1)^{s_x} \times m_x \times \beta^{e_x}$$

avec

$$m_x = x_0 \,.\, x_1 \; x_2 \; x_3 \; \cdots \; x_n$$

où $x_i \in \{0, 1, \ldots, \beta - 1\}$.

Pour des questions de précision, on exige que la mantisse soit normalisée, c'est-à-dire que son premier chiffre x_0 soit différent de 0. On a alors $m_x \in [1, \beta[$. Il faut alors un codage spécial pour le nombre 0.

Au début était le chaos. . .

Les représentations flottantes étaient pendant longtemps très différentes les unes des autres selon les constructeurs de processeurs.

machine	β	n	e_{min}	e_{max}
Cray 1	2	48	-8192	8191
	2	96	-8192	8191
DEC VAX	2	53	-1023	1023
	2	56	-127	127
HP 28 et 48G	10	12	-499	499
IBM 3090	16	6	-64	63
	16	14	-64	63
	16	28	-64	63

Problème : il n'était pas possible de faire raisonnablement des programmes et des bibliothèques numériques portables !

Autres exemples de problèmes

- IBM System/370 en FORTRAN on avait $\sqrt{-4} = 2$
- Sur certaines machines CDC et Cray on avait :

$$x - y \neq y - x$$

$$0.5 \times x \neq x/2.0$$

Avec ça, comment écrire des programmes numériquement corrects de façon simple ?

A l'époque, les constructeurs ne s'intéressent qu'aux formats de stockage des données et pas beaucoup aux propriétés mathématiques des unités de calcul. . .

Révolution de 1985 : la norme IEEE-754

Après de nombreuses années de travail, une norme fixe la représentation des données et le comportement des opérations de base en virgule flottante.

Cette norme fixe :
- les formats des données
- les valeurs spéciales
- les modes d'arrondi
- la précision des opérations de base
- les règles de conversion

En fait, il y a deux normes[1] :
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic* en 1985
- *ANSI/IEEE Standard for Radix-Independent Floating-Point Arithmetic* en 1987 (où $\beta = 2$ ou 10)

[1] ANSI : *American National Standards Institute*, IEEE : *Institute of Electrical and Electronics Engineers*

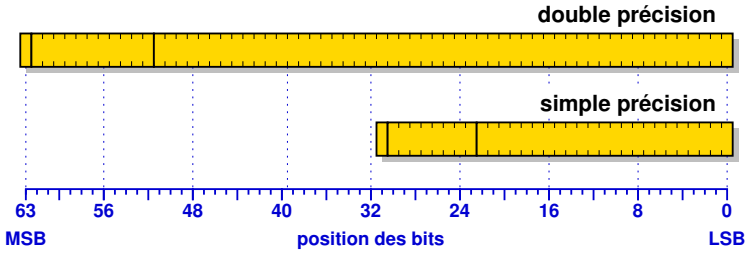
Objectifs de la norme IEEE-754

- permettre de faire des programmes portables
- rendre les programmes déterministes d'une machine à une autre
- conserver des propriétés mathématiques
- permettre l'arrondi correct
- permettre des conversions fiables
- faciliter la construction de preuves
- faciliter la gestion des exceptions
- faciliter les comparaisons (unicité de la représentation, sauf pour 0)
- permettre un support pour l'arithmétique d'intervalle

IEEE-754 : formats de base

En base $\beta = 2$, la normalisation de la mantisse implique que le premier bit est toujours un "1" qui n'est pas stocké physiquement, on parle de 1 implicite.

format	nombre de bits			
	total	signe	exposant	mantisse
double précision	64	1	11	$52+1$
simple précision	32	1	8	$23+1$



IEEE-754 : mantisse et fraction

La mantisse (normalisée) du nombre flottant x est représentée par $n+1$ bits :

$$m_x = 1 \,.\, \underbrace{x_1\ x_2\ x_3\ \cdots\ x_{n-1}\ x_n}_{f_x}$$

où les x_i sont des bits.

La partie fractionnaire de m_x est appelée fraction (de n bits) : f_x. On a alors :

$$m_x = 1 + f_x$$

On a aussi :

$$1 \leq m_x < 2$$

IEEE-754 : exposant

L'exposant e_x est une entier signé de k bits tel que :

$$e_{min} \leq e \leq e_{max}$$

Différentes représentations sont possibles : complément à 2, signe et magnitude, biaisée. C'est la représentation biaisée qui est choisie.

Ceci permet de faire les comparaisons entre flottants dans l'ordre lexicographique (sauf pour le signe s_x) et de représenter le nombre 0 avec $e_x = f_x = 0$.

L'exposant stocké physiquement est l'exposant biaisé e_b tel que :

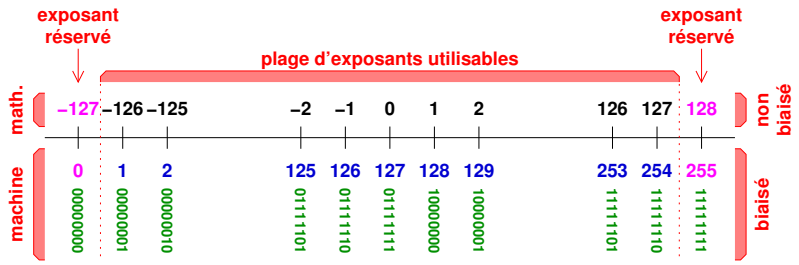
$$e_b = e + b$$

où b est le biais.

IEEE-754 : exposant (suite)

Les exposants non biaisés $e_{min}-1$ et $e_{max}+1$ (respectivement 0 et 2^k-1 en biaisé) sont réservés pour zéro, les dénormalisés et les valeurs spéciales.

format	taille k	biais b	non-biaisé e_{min}	non-biaisé e_{max}	biaisé e_{min}	biaisé e_{max}
SP	8	$127 \quad (=2^{8-1}-1)$	-126	127	1	254
DP	11	$1023 \quad (=2^{11-1}-1)$	-1022	1023	1	2046



IEEE-754 : zéro

Le nombre zéro est codé en mettant tous les bits de l'exposant et de la fraction à 0 dans le mot machine. Le bit de signe encore "libre" permet d'avoir deux représentations différentes du nombre zéro : -0 et $+0$.

Le fait que zéro soit signé est cohérent avec le fait qu'il y ait deux infinis distincts. On a alors :

$$\frac{1}{+0}=+\infty \qquad \text{et} \qquad \frac{1}{-0}=-\infty$$

La norme impose que le test $-0=+0$ retourne la valeur vrai.

En simple précision, on a donc :

-0	1	00000000	00000000000000000000000
$+0$	0	00000000	00000000000000000000000

IEEE-754 : valeurs spéciales

- Les infinis : $-\infty$ et $+\infty$
  Ils sont codés en utilisant le plus grand exposant possible et une fraction nulle. L'infini est signé.

$$e = e_{max} + 1 \qquad \text{et} \qquad f_x = 0$$

- Not a Number : `NaN`
  Permet de représenter le résultat d'une opération invalide telle que $0/0$, $\sqrt{-1}$ ou $0 \times \infty$. `NaN` est codé en utilisant le plus grand exposant possible et une fraction non-nulle. Les `NaN` se propagent dans les calculs.

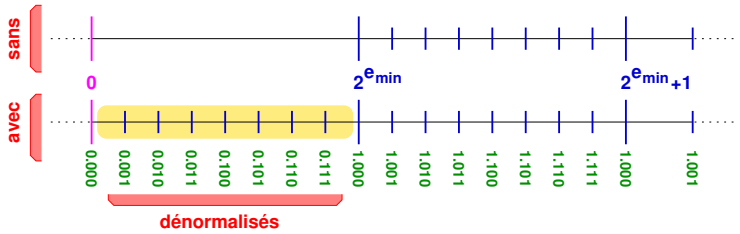
$$e = e_{max} + 1 \qquad \text{et} \qquad f_x \neq 0$$

En simple précision, on a donc :

$-\infty$	`1 11111111 00000000000000000000000`
$+\infty$	`0 11111111 00000000000000000000000`
`NaN`	`0 11111111 00000000000000000000100` (par exemple)

IEEE-754 : nombres dénormalisés

L'objectif des nombres dénormalisés est d'uniformiser la répartition des nombres représentables autour de 0. En effet, le 1 implicite dans la mantisse implique qu'il n'y a pas de nombre représentable entre 0 et $2^{e_{min}}$ alors qu'il y en a 2^n entre $2^{e_{min}}$ et $2^{e_{min}+1}$.



Les nombres dénormalisés s'écrivent (avec $e_b = 0$) :

$$x = (-1)^{s_x} \times (0.f_x) \times 2^{e_{min}}$$

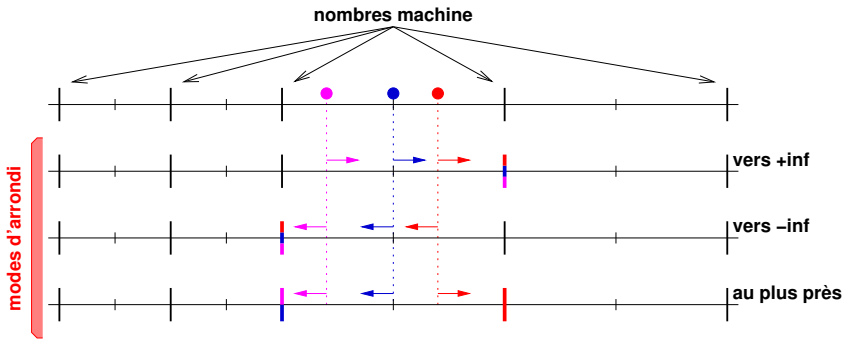
IEEE-754 : modes d'arrondis

Si a et b sont deux nombres exactement représentables en machine (ou nombres machine) alors le résultat d'une opération $r = a \circledcirc b$ n'est, en général, pas représentable en machine. Il faut arrondir le résultat. Par exemple, en base $\beta = 10$, le nombre $1/3$ n'est pas représentable avec un nombre fini de chiffres.

La norme propose 4 modes d'arrondi :

- arrondi vers $+\infty$ (ou par excès), noté $\triangle(x)$: retourne le plus petit nombre machine supérieur ou égal au résultat exact x
- arrondi vers $-\infty$ (ou par défaut), noté $\nabla(x)$: retourne le grand nombre machine inférieur ou égal au résultat exact x
- arrondi vers 0, noté $\mathcal{Z}(x)$: retourne $\triangle(x)$ pour les nombres négatifs et $\nabla(x)$ pour les positifs
- arrondi au plus près, noté $\circ(x)$: retourne le nombre machine le plus proche du résultat exact x (celui dont la mantisse se termine par un 0 pour le milieu de nombres machine consécutifs, on parle d'arrondi pair)

IEEE-754 : modes d'arrondis (suite)



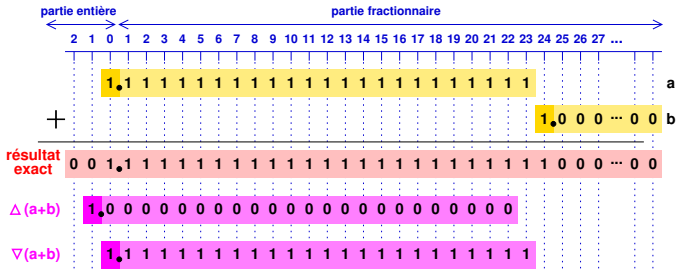
Propriété d'arrondi correct : soient a et b deux nombres machine, $\circledcirc$ une des opérations $(+, -, \times, \div)$ et $\diamond$ le mode d'arrondi choisi (parmi les 4 modes IEEE). Le résultat fourni lors du calcul de $(a \circledcirc_{th} b)$ doit être $\diamond(a \circledcirc b)$.

Le résultat retourné doit être celui obtenu par un calcule avec une précision infinie, puis arrondi. On a la même exigence pour la racine-carrée.

IEEE-754 : modes d'arrondi en C

```
#include <stdio.h>
#include "util-ieee.h"
#include <fenv.h> /* gestion de l'environnement flottant */

int main () {
  cfloat a, b, sup, sdown;

  /* a = 2 - 2^(-23)     et     b = 2^(-24)*/
  a.s.sig = 0;   a.s.exp = 0+127;     a.s.man = (2<<22)-1;
  b.s.sig = 0;   b.s.exp = -24+127;  b.s.man = 0;
  cfloat_print("               a",a);   cfloat_print("               b",b);

  fesetround(FE_UPWARD); /* passe en arrondi vers le haut */
  sup.f = a.f + b.f;
  cfloat_print("  (a+b) rnd up",sup);

  fesetround(FE_DOWNWARD); /* passe en arrondi vers le bas */
  sdown.f = a.f + b.f;
  cfloat_print("(a+b) rnd down",sdown);

  return 0; }
```


IEEE-754 : modes d'arrondi en C (suite)




```
  (a+b) rnd up = 2.00000000000000000000000000000000000000000e+00
  (a+b) rnd up = 0 128(    1)          0
  (a+b) rnd up = 0 128(    1) 1.00000000000000000000000
  (a+b) rnd up = 0x40000000

(a+b) rnd down = 1.99999988079071044921875000000000000000000e+00
(a+b) rnd down = 0 127(    0)    8388607
(a+b) rnd down = 0 127(    0) 1.11111111111111111111111
(a+b) rnd down = 0x3fffffff
```

IEEE-754 : conversions

Les conversions normalisées sont :

- flottant vers entier
- entier vers flottant
- flottant vers entier stocké dans un flottant
- entre flottants de différents formats
- entre formats binaire et décimal

Propriétés des doubles conversions imposées par la norme :

- binaire $x \longrightarrow$ décimal $x_{(10)} \longrightarrow$ binaire $x_{(2)}$:
  on retrouve le nombre initial, c'est-à-dire $x = x_{(2)}$, si $x_{(10)}$ a au moins 9 chiffres décimaux pour x en simple précision (17 en DP)
- décimal $y \longrightarrow$ binaire $y_{(2)} \longrightarrow$ décimal $y_{(10)}$:
  on retrouve le nombre initial, c'est-à-dire $y = y_{(10)}$, si y a au plus 6 chiffres décimaux et que $y_{(2)}$ en simple précision est converti en $y_{(10)}$ avec 6 chiffres décimaux (15 en DP)

IEEE-754 : comparaisons

La norme impose que l'opération de comparaison soit exacte et ne génére pas de dépassement de capacité.

Les comparaisons normalisées sont :

- égalité
- supérieur
- inférieur
- non-ordonné

Lors de ces comparaisons le signe de zéro n'est pas en compte.

Dans le cas de comparaisons impliquant un `NaN`, la comparaison retourne faux. Sauf dans le cas de l'égalité : si $x =$ `NaN` alors $x = x$ retourne faux[2] et $x \neq x$ retourne vrai.

[2] Ce qui permet de tester si une valeur est un NaN.

IEEE-754 : drapeaux ou exceptions

La norme précise qu'aucun calcul ne doit entraver le bon fonctionnement de la machine. Un mécanisme de drapeaux permet d'informer le système sur le comportement des opérations. La norme prévoit 5 drapeaux :

- opération invalide : le résultat par défaut est `NaN`
- dépassement de capacité vers $+\infty$ (*overflow*) : le résultat est soit $\pm\infty$ soit le plus grand nombre représentable suivant le signe du résultat exact et du mode d'arrondi
- depassement de capacité vers 0 (*underflow*) : le résultat est soit ± 0 soit un dénormalisé
- division par zéro : le résultat est $\pm\infty$
- résultat inexact : levé lorsque que le résultat d'une opération n'est pas exact

Ces drapeaux, une fois levés, le restent pendant tout le calcul jusqu'à une remise à zéro volontaire (*sticky flags*). Ils peuvent être lus et écrits par l'utilisateur.

IEEE-754 : dynamique de la représentation

La dynamique d'une représentation est le rapport entre le plus grand nombre et le plus petit nombre représentables et strictement positifs.

- en virgule fixe sur n bits, on a :

$$D_{fixe} = \frac{2^n - 1}{1}$$

- en virgule flottante (e sur k bits et m sur n bits), on a :

$$D_{flottant} = \frac{m_{max} \times 2^{e_{max}}}{m_{min} \times 2^{e_{min}-1}} = \frac{(2 - 2^{-n}) \times 2^{2^{k-1}-1}}{(2^{-n}) \times 2^{-2^{k-1}+2}}$$

Pour 32 bits on a : $D_{fixe} = 4.29 \times 10^9$ et $D_{flottant} = 2.43 \times 10^{85}$.

IEEE-754 : autres formats

format	taille totale	taille exposant	taille mantisse
simple étendu	≥ 32	≥ 11	n.c.
double étendu	≥ 64	≥ 15	n.c.
double étendu PC	80	15	63
quad (Sun)	128	15	112

Dans les formats étendus, il n'y a pas de bit implicite pour la mantisse, il doit être stocké dans le mot machine.

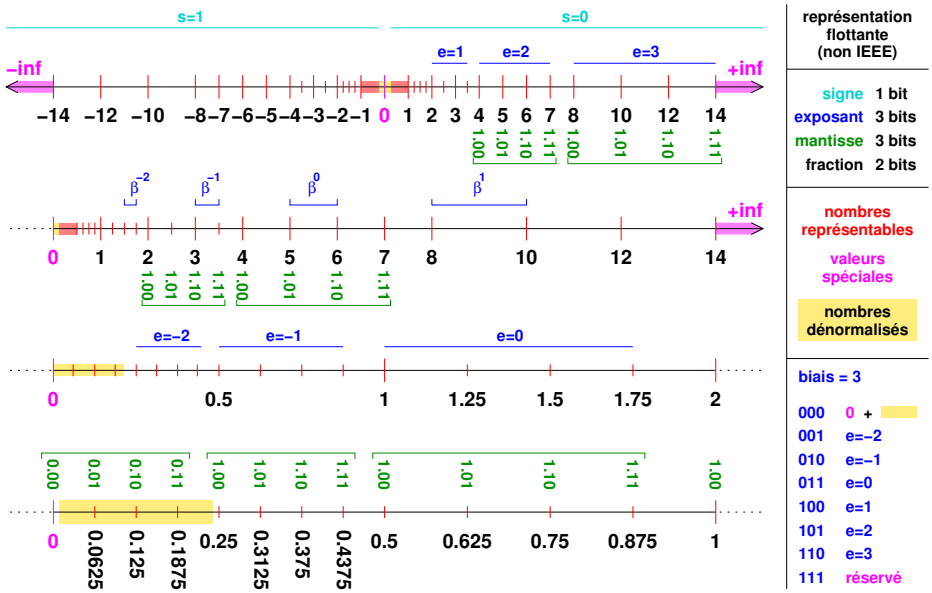
IEEE-754 : résumé

	exposant	fraction	valeur
normalisés	$e_{min} \leq e \leq e_{max}$	$f \geq 0$	$\pm(1.f) \times 2^e$
dénormalisés	$e = e_{min} - 1$	$f > 0$	$\pm(0.f) \times 2^{e_{min}}$
zéro (signé)	$e = e_{min} - 1$	$f = 0$	± 0
infinis	$e = e_{max} + 1$	$f = 0$	$\pm\infty$
Not a Number	$e = e_{max} + 1$	$f > 0$	`NaN`

format	# bits total	k	n	e_{min}	e_{max}	b
simple précision	32	8	23	-126	127	127
double précision	64	11	52	-1022	1023	1023

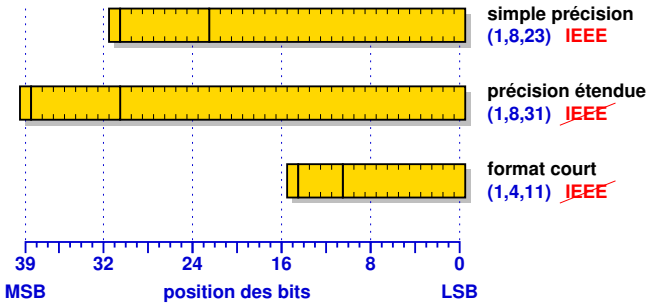
valeur	simple précision	double précision
+ grand normalisé > 0	$3.40282347 \times 10^{38}$	$1.7976931348623157 \times 10^{308}$
+ petit normalisé > 0	$1.17549435 \times 10^{-38}$	$2.2250738585072014 \times 10^{-308}$
+ grand dénormalisé > 0	$1.17549421 \times 10^{-38}$	$2.2250738585072009 \times 10^{-308}$
+ petit dénormalisé > 0	$1.40129846 \times 10^{-45}$	$4.9406564584124654 \times 10^{-324}$

Exemple de représentation simplifiée



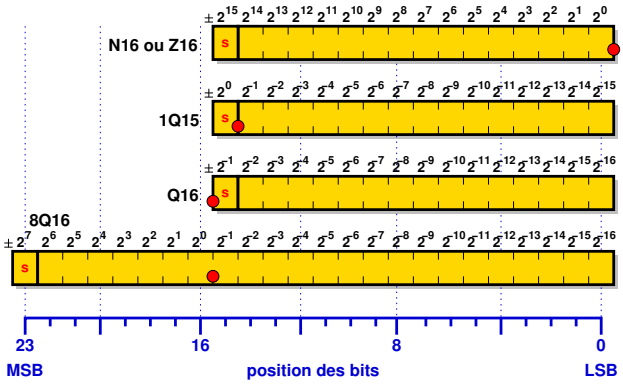
Représentations flottantes non-standards

Exemple du processeur DSP (*digital signal processor*) SHARC 21160 d'Analog Devices où il y a plusieurs formats flottants non-standards (et seulement un compatible IEEE-754 simple précision).



Alternative aux flottants : la virgule fixe

Dans bon nombre de processeurs DSP, on trouve un support matériel très efficace (en vitesse et consommation d'énergie) pour le calcul sur les *réels* à l'aide de multiples formats en virgule fixe (16, 24 ou 32 bits).



Mais la qualité numérique et la portablilité sont moindres qu'en IEEE-754

Références sur les flottants IEEE-754

- Documentation de W. Kahan (le "père" de la norme)
  `http://www.cs.berkeley.edu/~wkahan/ieee754status/`
- Documentation générale et scripts Java pour faire des conversions
  `http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html`
- Version web de l'article de D. Goldberg dans *ACM Computing Surveys*
  `http://docs.sun.com/source/806-3568/ncg_goldberg.html`
- Documentation du Centre Charles Hermite (Nancy)
  `http://cch.loria.fr/documentation/IEEE754/`

Multiplication-addition fusionnée (FMA)

Le FMA (pour *fused multiply and add*) existe depuis de nombreuses années dans les DSP et arrive de plus en plus dans les processeurs généralistes. Cette opération effectue le calcul suivant en une seule instruction (au lieu de 2 sans unité FMA).

$$r = (a + b \times c)$$

Pour le moment le comportement de cette opération n'est pas normalisé en IEEE-754. En pratique les processeurs généralistes qui possède une unité FMA retournent le meilleur résultat possible : l'arrondi du résultat théorique (arrondi correct de $(a + b \times c)$) en effectuant un seul arrondi.

Exemple d'utilisation : évaluation de polynômes

$$p(x) = p_0 + (p_1 + (p_2 + (p_3 + p_4\ x)\ x)\ x)\ x$$

Exemples de propriétés vraies en machine

L'addition flottante $\oplus$ et la multiplication flottante $\otimes$ sont commutatives mais pas associatives (dépassements de capacité). La multiplication flottante n'est pas distributive par rapport à l'addition.

Si aucun dépassement de capacité vers l'infini ou vers zéro ne se produit pendant les calculs, les propriétés suviantes sont vérifiées avec des nombres flottants et des opérations flottantes IEEE.

$$\begin{aligned} x \oplus 0 &= x \ominus 0 = x \\ x \otimes 1 &= x \oslash 1 = x \\ x \otimes -1 &= x \oslash -1 = -x \\ 2 \otimes x &= x \oplus x = 2x \\ 0.5 \otimes x &= x \oslash 2 = x/2 \end{aligned}$$

$$\begin{aligned} \circ(\sqrt{x}) \otimes \circ(\sqrt{x}) &= x \\ \circ(\sqrt{x}) &\geq 0 \quad \text{si} \quad x \geq 0 \\ \circ(\sqrt{-0}) &= -0 \end{aligned}$$

Exemples de propriétés vraies en machine (suite)

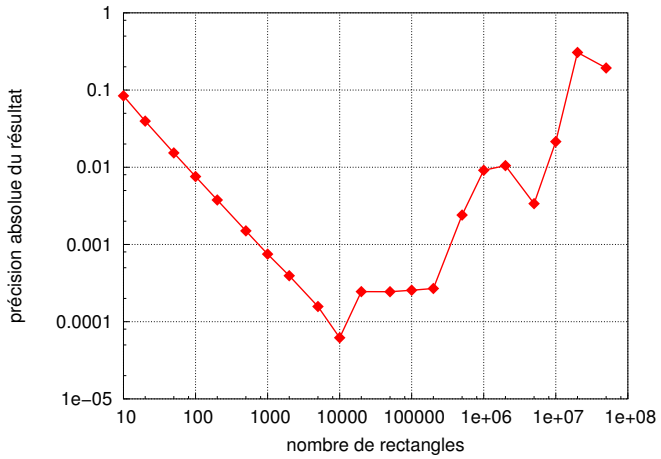
Grâce à la norme IEEE, en l'absence de dépassement de capacité (vers $+\infty$ ou vers 0) et de division par 0, avec x et y deux nombres machine on a :

$$-1 \quad \leq \quad \frac{x}{\sqrt{x^2+y^2}} \quad \leq \quad 1$$

même après 5 erreurs d'arrondi.

Problème d'intégration numérique

A l'aide de la méthode des rectangles, essayons de calculer $\int_1^2 \frac{dx}{x}$ (le résultat théorique est $\ln(2)$), et ce pour plusieurs nombres de rectangles :



Précision et erreur d'arrondi

A chaque arrondi, il est possible de perdre un peu de précision, on parle d'erreur d'arrondi. Même si une opération isolée retourne le meilleur résultat possible (l'arrondi du résultat exact), une suite de calculs peut conduire à d'importantes erreurs du fait du cumul des erreurs d'arrondi.

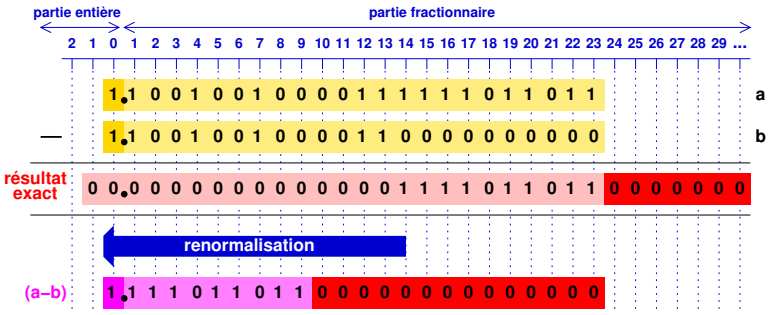
Précision en nombre de bits justes ($p_{bits} = -\log_2(|err_{absolue}|)$) :

$$\begin{aligned} a &= 1.110\,000\,000 = 1.75 \\ a' &= 1.101\,111\,111 = 1.748046875 \\ |a - a'| &= 0.000\,000\,001 = 0.00193125 \\ p_{bits} &= 9 \end{aligned}$$

Si a est la valeur exacte, alors a' représente a avec un 1 bit faux (et pas 8). En effet, $|a - a'| = 2^{-9}$, où 2^{-9} est le poids du dernier bit de a.

Phénomène de cancellation (ou élimination)

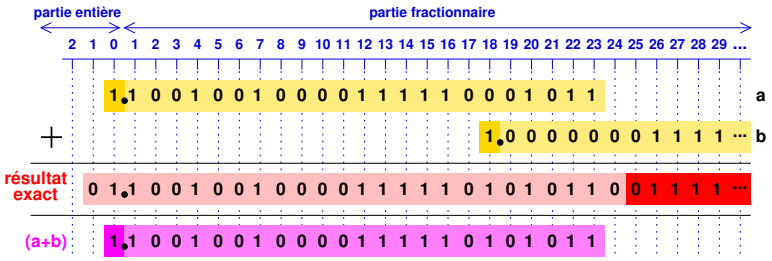
Se produit lors de la soustraction de deux nombres très proches.



L'opération $(a - b)$ est "exacte" car les opérandes sont supposées exactes. Mais si les opérandes sont elles-même des résultats de calculs avec des erreurs d'arrondi, les 0 ajoutés à droite (partie rouge foncée) sont faux. La cancellation est dite catastrophique quand il n'y a presque plus de chiffres significatifs.

Phénomène d'absorption

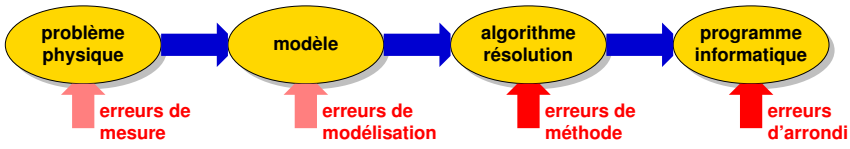
Se produit lors de l'addition de deux nombres ayant des ordres de grandeur très différents où l'on peut "perdre" le plus petit.



On parle même d'absorption catastrophique dans certains cas. Exemple : en simple précision, avec $a = 2^{30}$ et $b = 2^{-30}$ on a alors :

$$a \oplus b = 2^{30} \quad \text{et donc} \quad (a \oplus b) \ominus a = 0$$

Les différentes sources d'erreur



A notre niveau, on peut "seulement" travailler pour proposer des algorithmes numériques avec de plus petites erreurs de méthode et des implantations logicielles qui maîtrisent mieux les problèmes d'arrondi.

En pratique, pour obtenir de bons résultats, il faut concevoir des algorithmes en prenant en compte dès le début les erreurs d'arrondi.

Explosion du vol Ariane 501

Le 4 juin 1996, lors de son premier vol, la fusée européenne Ariane 5 explose 30 secondes après son décollage causant la perte de la fusée et de son chargement estimé à 500 M$.

Après deux semaines d'enquête, un problème est trouvé dans le système de référence inertiel. La vitesse horizontale de la fusée par rapport au sol était calculée sur des flottants 64 bits. Dans le programme du calculateur de bord, il y avait une conversion de cette valeur flottante 64 bits vers un entier signé 16 bits. Malheureusement, rien n'était fait pour tester que cette conversion était bien possible mathématiquement (sans dépassement de capacité). . .

Un petit programme rigolo

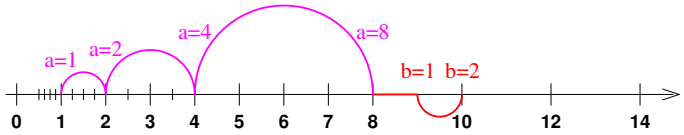
Que fait le programme suivant, dû à Gentleman [3] ?

```
#include <stdio.h>

int main() {
  float a = 1.0, b = 1.0;

  while ( (((a + 1.0) - a) - 1.0) == 0.0)
     a = 2.0 * a;

  while ( (((a + b) - a) - b) != 0.0)
    b = b + 1.0;

  printf("Gentleman : %f \n", b);

  return 0;
}
```

[3] W. M. Gentleman, More on algorithms that reveal properties of floating point arithmetic units, *Communications of the ACM*, vol. 17, n. 5, 1974.

Un petit programme rigolo (suite)

Dans le cas de notre représentation de test $(1, 3, 3)$, on a :

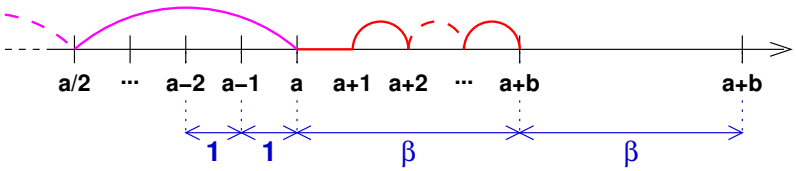


a	b	tests effectués
1.0	1.0	$((a+1.0)-a)-1.0 = ((2.0)-a)-1.0 = (1.0)-1.0 = 0$
2.0	1.0	$((a+1.0)-a)-1.0 = ((3.0)-a)-1.0 = (1.0)-1.0 = 0$
4.0	1.0	$((a+1.0)-a)-1.0 = ((5.0)-a)-1.0 = (1.0)-1.0 = 0$
8.0	1.0	$((a+1.0)-a)-1.0 = ((\underbrace{8.0}_{round(9)})-a)-1.0 = (0.0)-1.0 = -1.0$
8.0	1.0	$((a+b)-a)-b = ((\underbrace{8.0}_{round(9)})-a)-b = (0.0)-b = -1.0$
8.0	2.0	$((a+b)-a)-b = ((10.0)-a)-1.0 = (2.0)-b = 0.0$

Un petit programme rigolo (fin)

Le programme de Gentleman retourne la base utilisée par les unités de calcul flottant du processeur (2 dans le cas de mon PC avec un Pentium III).

Pourquoi ?



- première boucle : recherche de a tel que $a+1$ ne soit pas représentable
- deuxième boucle : recherche de b tel que $a+b$ soit représentable
- le résultat est $\beta = b$ la base interne de l'unité flottante

Partie 3

Addition et représentations redondantes

Points abordés dans cette partie

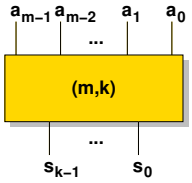
- Addition classique
- Addition rapide
- Représentations redondantes
- Addition redondante

Cellules de base pour l'addition

En plus des portes logiques précédentes, nous allons utiliser des nouvelles portes présentant une propriété arithmétique bien utile : le comptage de 1.

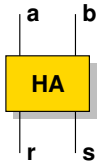
Un compteur (m, k) est une cellule, élémentaire ou non, qui compte le nombre de 1 présents sur ses m entrées et donne le résultat en numération simple de position sur k bits en sortie.

$$\sum_{i=0}^{m-1} a_i = \sum_{j=0}^{k-1} s_j 2^j$$



La cellule demi-additionneur (*half-adder* ou HA) est un compteur (2,2) tandis que la cellule d'addition complète (*full-adder* ou FA) est un compteur (3,2). Ces deux portes sont largement utilisées dans les opérateurs arithmétiques.

La cellule HA



a	b	r	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

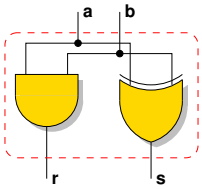
Équation arithmétique :

$$2r + s = a + b$$

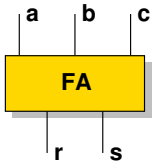
Équations logiques :

$$\begin{aligned} s &= a \oplus b \\ r &= ab \end{aligned}$$

Réalisation pratique du HA :



La cellule FA



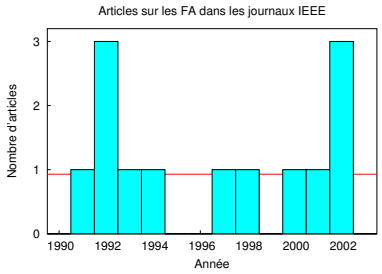
a	b	c	r	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Équation arithmétique :

$$2r + s = a + b + c$$

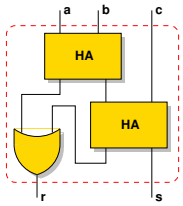
Équations logiques :

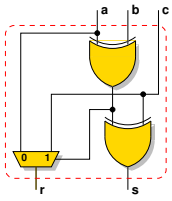
$$\begin{aligned} s &= a \oplus b \oplus c \\ r &= ab + ac + bc \end{aligned}$$

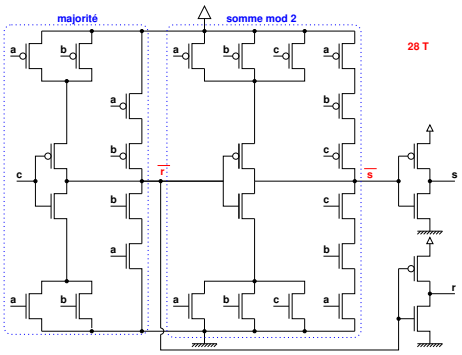


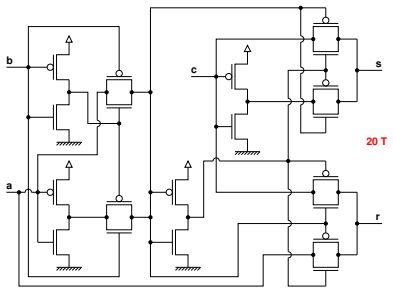
Il existe de nombreuses réalisations pratiques de la cellule FA.

Quelques implantations possibles de la cellule FA



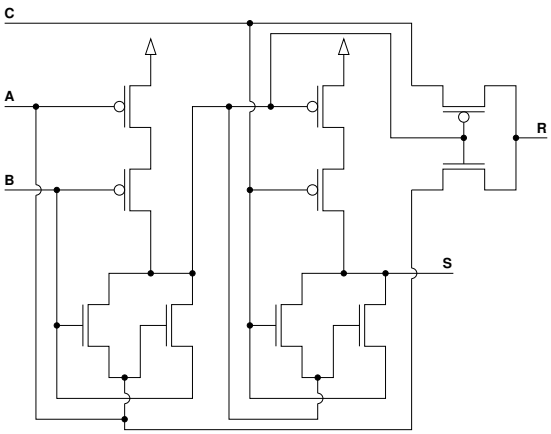






La cellule FA du jour

Solution en 10 transistors[4] :

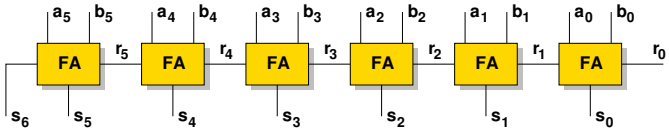


A	B	C	R	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0_w	1
0	1	1	1	0
1	0	0	0_w	1
1	0	1	1_w	0
1	1	0	1	0
1	1	1	1_w	1_w

[4] H. T. Bui, Y. Wang et Y. Jiang. *Design and analysis of low-power 10-transistor full adders using novel XOR–XNOR gates*. IEEE Trans. CaS, jan. 2002.

Additionneur séquentiel

C'est l'additionneur le plus simple. Il est composé de n cellules FA connectées en série.



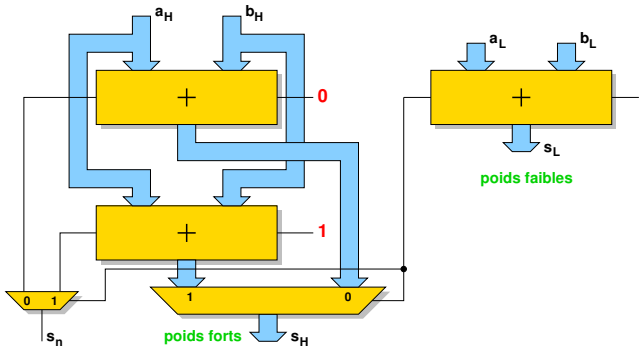
Dans la littérature, on le trouve sous le nom de *Ripple-Carry Adder* (RCA) ou parfois de *Carry-Propagate Adder* (CPA)[5].

	complexité
délai	$O(n)$
surface	$O(n)$

[5] Attention : CPA peut aussi désigner un additionneur non redondant (propage mais ne conserve pas).

Additionneur à sélection de retenue

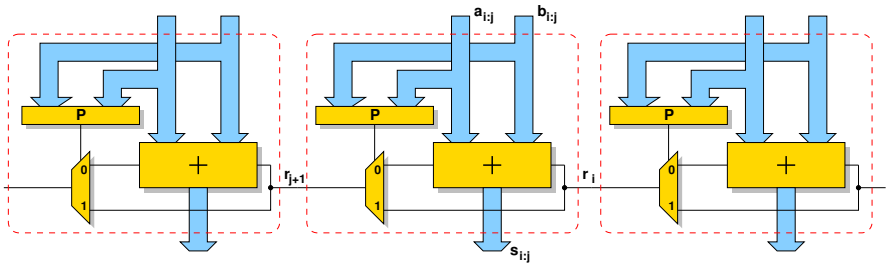
Idée : couper en deux et calculer le bloc des poids forts avec les deux retenues entrantes possibles et sélectionner la bonne sortie avec la retenue sortante des poids faibles (*Carry-Select Adder* ou *Conditional-Sum Adder*).



Version récursive $\longrightarrow$ délai en $O(\log n)$ (mais sortance non bornée).

Additionneur à retenue bondissante

Idée : découper en blocs où chaque bloc permet de détecter rapidement une propagation sur tout le bloc (*Carry-Skip Adder*).



Questions :

- blocs de même taille ?
- taille optimale des blocs ?

Cas des blocs de même taille

Le pire cas d'un additionneur de n bits découpé en blocs de k bits se produit lorsque l'on doit propager du bit 1 jusqu'au bit $n-2$ (générations de retenues aux bits 0 et $n-1$ et propagation au milieu). Soit τ_1 le temps de propagation sur un 1 bit et τ_2 le temps de saut d'un bloc de k bits ($\tau_2 < \tau_1$).

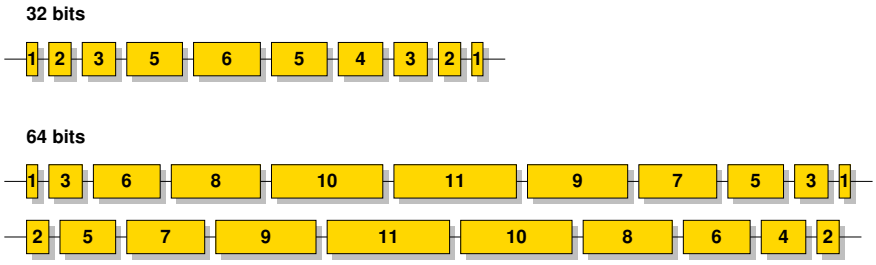
$$\begin{aligned} T(k) &= 2(k-1)\tau_1 + (\frac{n}{k} - 2)\tau_2 \\ T'(k) &= 2\tau_1 - \frac{n\tau_2}{k^2} \\ T''(k) &= \frac{2n\tau_2}{k^3} \end{aligned}$$

$$k_{opt} = \sqrt{\frac{n\tau_2}{2\tau_1}} \quad \longrightarrow \quad T(k_{opt}) = O(\sqrt{n})$$

Cas des blocs de tailles variables

Le problème est compliqué dans le cas général, mais de nombreuses solutions (heuristiques) ont été proposées pour les cas se présentant en pratique.

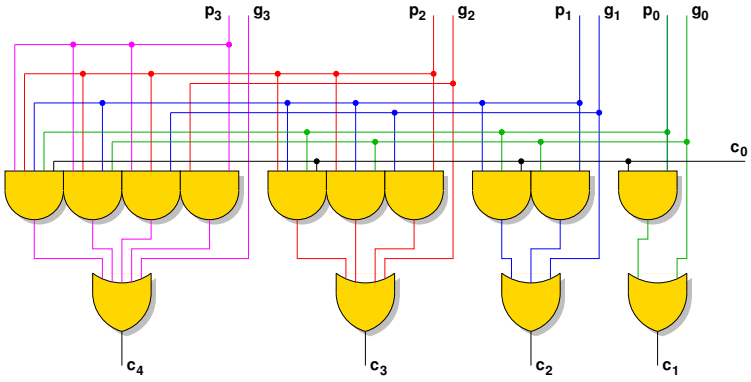
Exemple de solutions dans *A Simple Strategy for Optimized Design of One-Level Carry-Skip Adders*. M. Alioto et G. Palumbo. IEEE Trans. CaS I, jan. 03.



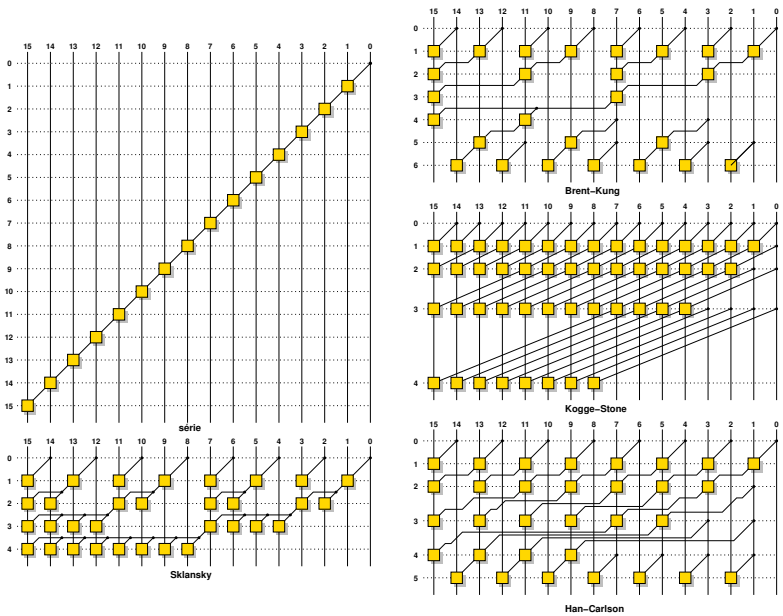
Calcul des retenues pour un CLA 4 bits

CLA = *carry lookahead adder*

$$c_1 = g_0 + p_0c_0 \qquad c_3 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$$
$$c_2 = g_1 + p_1g_0 + p_1p_0c_0 \qquad c_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0$$



Quelques additionneurs à préfixe parallèle



représentations redondantes

Pour aller encore plus vite, on va "tricher" en conservant les retenues. Cela n'a un sens que si l'on effectue plusieurs additions successivement.

En 1961, Avizienis a suggéré de représenter les nombres en base β par l'ensemble de chiffres $\{-\alpha, -\alpha+1, \ldots, 0, \ldots, \alpha-1, \alpha\}$ au lieu des chiffres de $\{0, 1, 2, \ldots, \beta-1\}$ avec $\alpha \leq \beta - 1$.

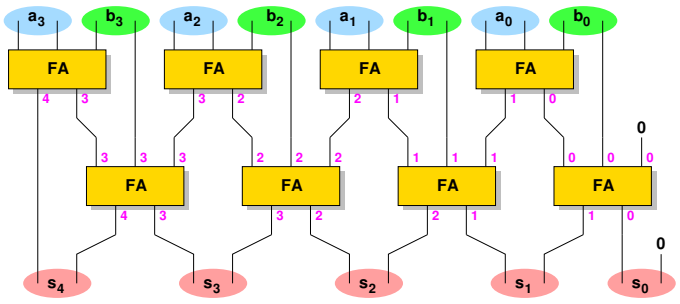
Dans ce système, si $2\alpha + 1 > \beta$ certains nombres ont plusieurs écritures possibles. Par exemple, le nombre 2345 (dans le système usuel) peut s'écrire en base 10 avec l'ensemble de chiffres $\{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$ selon les codages 2345, 235(-5) ou 24(-5)(-5). On dit alors que le système est redondant.

L'intérêt des systèmes de représentation redondants est qu'il existe dans ces systèmes des algorithmes permettant d'effectuer des additions de façon totalement parallèle, c'est-à-dire sans propagation de retenues.

Additionneur *carry-save*

On représente le nombre A en base 2 avec les chiffres $a_i \in \{0, 1, 2\}$ sur deux fils tels que $a_i = a_{i,c} + a_{i,s}$ où $a_{i,c} \in \{0, 1\}$ et $a_{i,s} \in \{0, 1\}$.

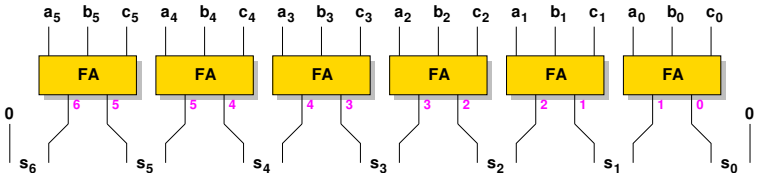
$$A = \sum_{i=0}^{n-1} a_i 2^i = \sum_{i=0}^{n-1} (a_{i,c} + a_{i,s}) 2^i$$



Toutes les sommes sont obtenues dans le délai de 2 cellules FA.

Addition *carry-save* de $k \geq 3$ nombres standards

Soient A, B et C trois nombres en notation non-redondante. On obtient leur somme S en représentation *carry-save* avec l'opérateur suivant ($k = 3$).



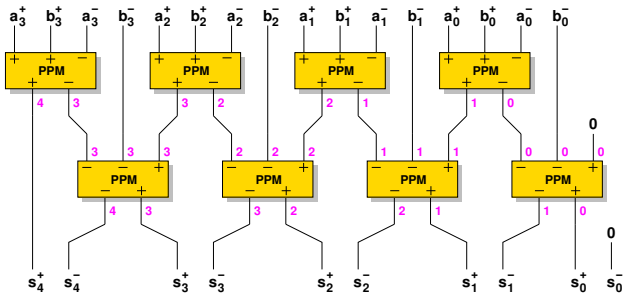
Plus généralement, un arbre *carry-save* à h niveaux permet de réduire au plus $n(h)$ entrées non-redondantes selon le tableau ci-dessous. On a $n(h) = \lfloor 3n(h-1)/2 \rfloor$ et $n(0) = 2$.

h	1	2	3	4	5	6	7	8	9	10	11
$n(h)$	3	4	6	9	13	19	28	42	63	94	141

Additionneur *borrow-save*

On représente le nombre A en base 2 avec les chiffres $a_i \in \{-1, 0, 1\}$ sur deux fils tels $a_i = a_i^+ - a_i^-$ où $a_i^+ \in \{0, 1\}$ et $a_i^- \in \{0, 1\}$.

$$A = \sum_{i=0}^{n-1} a_i 2^i = \sum_{i=0}^{n-1} (a_i^+ - a_i^-) 2^i$$



Toutes les sommes sont obtenues dans le délai de 2 cellules PPM.

Cellule PPM

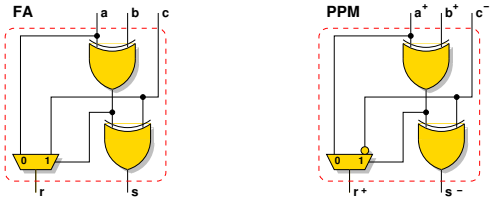
a^+	b^+	c^-	r^+	s^-
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Équation arithmétique :

$$2r^+ - s^- = a^+ + b^+ - c^-$$

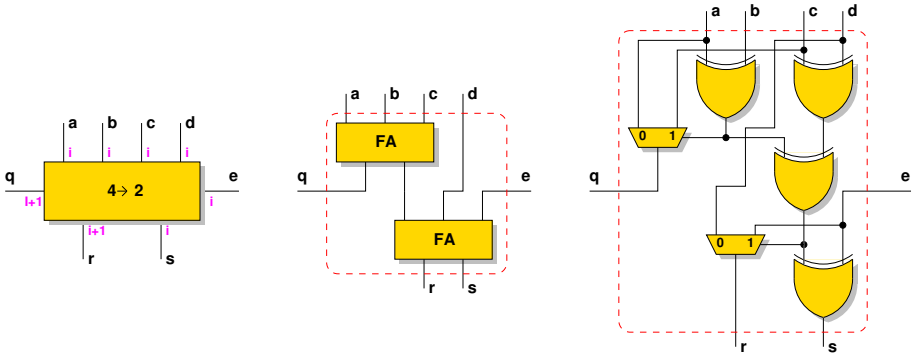
Équations logiques :

$$\begin{aligned} s &= a^+ \oplus b^+ \oplus c^- \\ r &= a^+ b^+ + a^+ \overline{c^-} + b^+ \overline{c^-} \end{aligned}$$



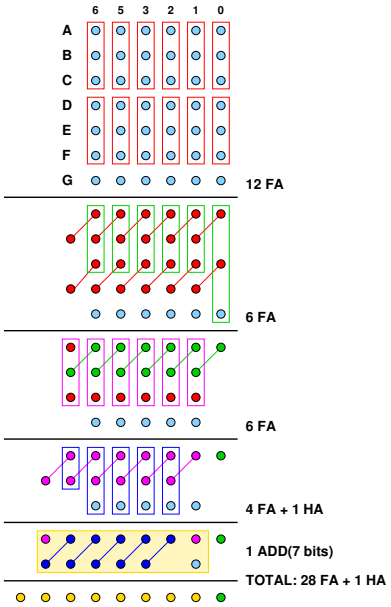
Cellule 4 donne 2

Equation arithmétique : $a + b + c + d + e = 2(r + q) + s$

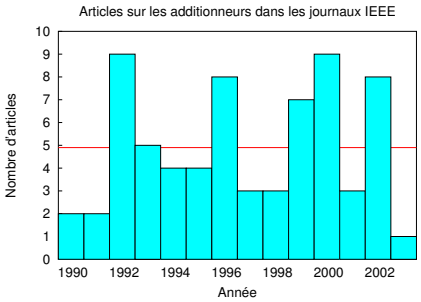


Intérêt : faire des arbres plus réguliers qu'avec la cellule FA (3 donne 2).

Additionneurs multi-opérandes : réduction en points



La recherche sur les additionneurs



- 1994 : NEC, CMOS 0.4 μm et 3.3 V
  ALU basée sur un CLA 32 bits à 500 MHz.
- 2002 : Intel, CMOS 0.13 μm et 1.5 V
  ALU basée sur un Han-Carlson 32 bits à 5 GHz.

Partie 4

Algorithmes de division

Points abordés dans cette partie

- Division simple par additions–décalages
- Division SRT
- Méthode de Newton
- Extensions au calcul de la racine carrée
- Fréquence d'utilisation de la division en machine

Division à la main

On cherche à trouver le quotient q de la division du dividende x par le diviseur d (le reste est r). On a : $x = q \times d + r$.

Exemple : calcul de $123/234$, (résultat exact $\frac{123}{234} = 0.5256410256\ldots$).

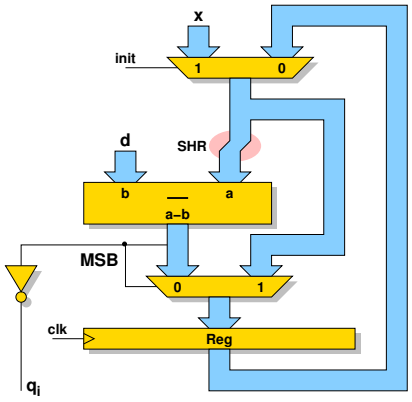
1	2	3	0				2	3	4	
		6	0	0						
		1	3	2	0		.	5	2	5	6	4	···
			1	5	0	0							
					9	6	0						
						2	4						

1	x	234	=	234
2	x	234	=	468
3	x	234	=	702
4	x	234	=	936
5	x	234	=	1170
6	x	234	=	1404
7	x	234	=	1638
8	x	234	=	1872
9	x	234	=	2106
10	x	234	=	2340

On effectue donc l'itération : $x^{(i+1)} = 10x^{(i)} - q_{i+1}d$

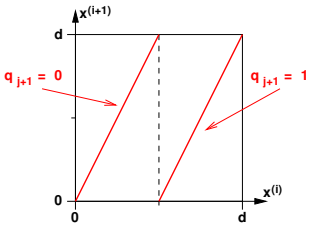
Division restaurante

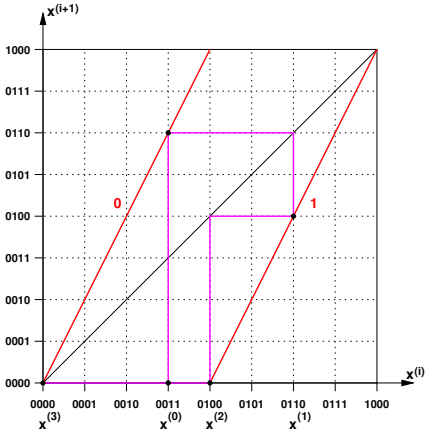
```
1  for i from 1 to n do
2      x ⟵ 2x
3      x ⟵ x − d
4      if x ≥ 0 then
5          q_i ⟵ 1
6      else
7          q_i ⟵ 0
8          x ⟵ x + d
```



Le caractère *restaurant* de cet algorithme vient de l'annulation de l'effet de la ligne 3 par la ligne 8 dans certains cas.

Diagramme de Robertson de la division restaurante





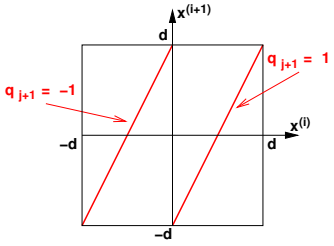
x = 11
d = 1000

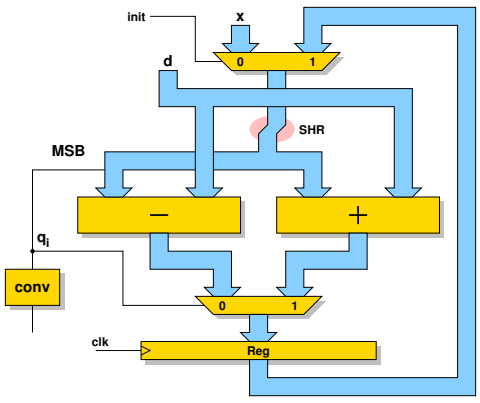
$x^{(0)}$ = 0011	q_0 = 0
$x^{(1)}$ = 0110	q_1 = 1
$x^{(2)}$ = 0100	q_2 = 1
$x^{(3)}$ = 0000	q_3 = 0

$\frac{3}{8} = 0.375$

Division non restaurante

```
1  for i from 1 to n do
2      x ⟵ 2x
3      if x ≥ 0 then
4          x ⟵ x − d
5          q_i ⟵ 1
6      else
7          x ⟵ x + d
8          q_i ⟵ −1
```



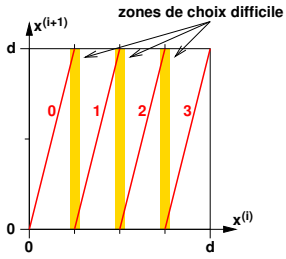


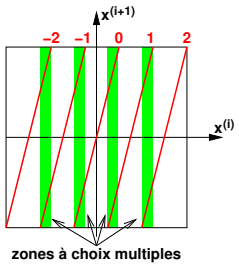
La conversion de $\{-1, 1\}$ vers $\{0, 1\}$ peut se faire à la volée (MSDF) avec un petit opérateur.

Comment aller plus vite ?

Idée : faire des itérations avec des q_i dans une base plus grande.

Problème : faire des comparaisons précises avec plusieurs multiples du diviseur.





Solution : utiliser une représentation redondante pour le quotient.
↪ faire des comparaisons approchées

Division SRT

La méthode SRT proposée par Sweeney, Robertson et Tocher en 1958 est basée sur :

- une représentation redondante des chiffres du quotient en base β (pour simplifier le choix des q_i).
- une représentation redondante des restes partiels (pour accélérer la soustraction, en *borrow-save* par exemple).
- une table permettant de déduire q_{i+1} à partir de quelques bits de poids forts de d et de $x^{(i)}$ (après conversion en non-redondant).

1 $d' \longleftarrow \mathrm{trunc}(d, k_d, \mathrm{MSB})$
2 **for** i from 1 **to** $n/\log_2 \beta$ **do**
3 $\quad x'^{(i)} \longleftarrow \mathrm{trunc}(x^{(i)}, k_x, \mathrm{MSB})$
4 $\quad q_{i+1} \longleftarrow \mathrm{T}(d', \mathrm{conv}(x'^{(i)}))$
5 $\quad x^{(i+1)} \longleftarrow \beta x^{(i)} - q_{i+1} \times d$

Architecture générale d'un diviseur SRT

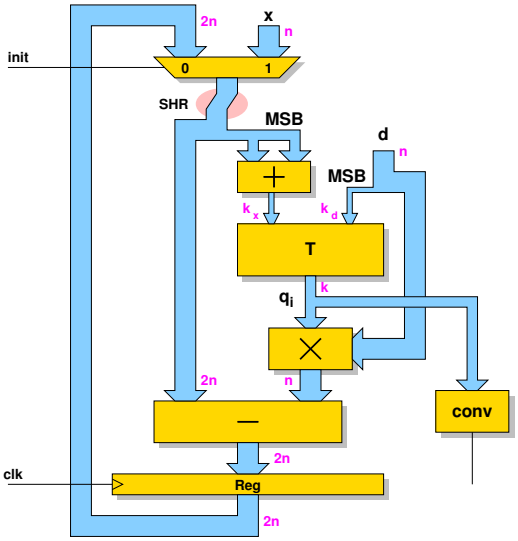
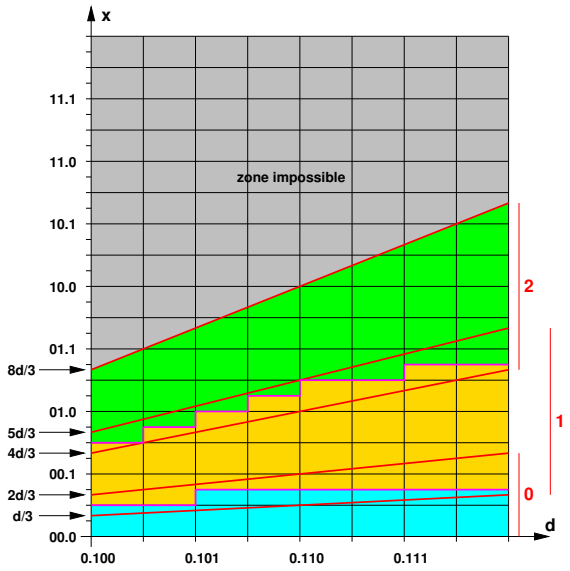


Table pour un diviseur SRT

Diagramme reste-diviseur :

- base $\beta = 4$
- $q_i \in \{-2, -1, 0, 1, 2\}$
- entrée :
  - d sur 3 bits
  - x sur 5 bits

Remarque : Le diagramme est symétrique par rapport à l'axe d.



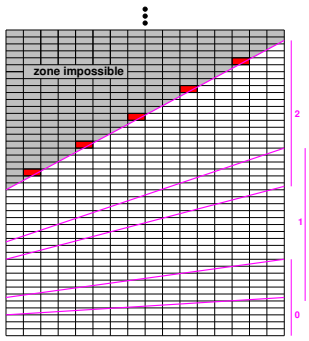
Le bug de la division du Pentium[6]

Exemple de problème en 1994 :

$$\begin{aligned} \frac{4195835.0}{3145727.0} &= 1.333739068902\ldots \quad \text{sur le pentium} \\ &= 1.333820449136\ldots \quad \text{exact} \end{aligned}$$

Diviseur du Pentium : SRT base 4, chiffres du quotient $\{-2,-1,0,1,2\}$, restes partiels en *carry-save*, table avec 5 bits d'entrée pour d et 7 bits pour $x^{(i)}$.

Problème : 5 cases fausses dans la table



[6] RR 95-06 de J.-M. Muller sur ce sujet à `http://www.ens-lyon.fr/LIP/`.

Extensions à la racine carrée

On peut calculer des racines carrées avec un algorithme à additions-décalages proche de celui pour la divison.

Exemple : on cherche $r = \sqrt{c}$ avec $x^{(0)} = c - 1$ et

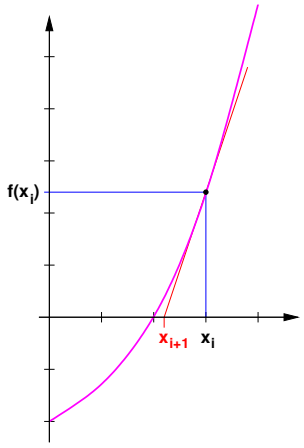
1 **for** i from 1 **to** $n/\log_2 \beta$ **do**
2 $\quad x'^{(i)} \longleftarrow \mathrm{trunc}(x^{(i)}, k_x, \mathrm{MSB})$
3 $\quad r'^{(i)} \longleftarrow \mathrm{trunc}(r^{(i)}, k_r, \mathrm{MSB})$
4 $\quad r_{i+1} \longleftarrow \mathrm{T}(\mathrm{conv}(r'^{(i)}), \mathrm{conv}(x'^{(i)}))$
5 $\quad r^{(i+1)} \longleftarrow r^{(i)} + r_{i+1}\beta^{-i-1}$
6 $\quad x^{(i+1)} \longleftarrow \beta x^{(i)} - 2r^{(i)}r_{i+1} - r_{i+1}^2\beta^{-i-1}$

Méthode de Newton

On cherche une racine de l'équation $f(x) = 0$ (où f est supposée continument dérivable) en utilisant la suite (x_i) définie par :

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Si x_0 est suffisament proche d'une racine simple α de f, alors la suite (x_i) converge quadratiquement vers α (le nombre de chiffres significatifs double à chaque étape).



$f(x) = \frac{x^2}{2} - 2$

i	0	1	2	3	4
x_i	3	2.166666667	2.006410256	2.000010240	2.000000000

Méthode de Newton pour la division

Pour trouver le quotient $q = a/d$ on va procéder en 2 étapes :

1. calcul de $t = 1/d$ à l'aide de la fonction $f(x) = \frac{1}{x} - d$. On doit donc calculer l'itération suivante :

$$\begin{aligned} x_{i+1} &= x_i - \frac{\frac{1}{x_i} - d}{-\frac{1}{{x_i}^2}} \\ &= x_i + x_i - d{x_i}^2 \\ &= x_i(2 - d{x_i}^2) \end{aligned}$$

   Le coût de chaque itération est de 2 multiplications et 1 addition.

   La valeur x_0 est obtenue par une lecture dans une petite table.

2. calcul de $q = t \times a$

Exemple de l'Itanium

Utilisation du multiplieur–accumulateur flottant sur des registres de 82 bits. Initialisation de la méthode de Newton par une lecture de table qui donne une approximation de $1/d$ à 8.886 bits près.

Exemple d'algorithme pour la simple précision (mantisse de 23 bits) :

$$
\begin{array}{ll}
1 & y_0 \longleftarrow \mathrm{T}(d) \\
2 & q_0 \longleftarrow (a \times y_0)_{rn} \\
3 & e_0 \longleftarrow (1 - b \times y_0)_{rn} \\
4 & q_1 \longleftarrow (q_0 + e_0 \times q_0)_{rn} \\
5 & e_1 \longleftarrow (e_0 \times e_0)_{rn} \\
6 & q_2 \longleftarrow (q_1 + e_1 q_1)_{rn} \\
7 & e_2 \longleftarrow (e_1 \times e_1)_{rn} \\
8 & q_3 \longleftarrow (q_2 + e_2 \times q_2)_{rn} \\
9 & q_3' \longleftarrow \mathrm{round}(q_3)
\end{array}
$$

Méthode de Newton pour la racine carrée

Première idée : utiliser Newton avec $f(x) = x^2 - c$, on a alors :

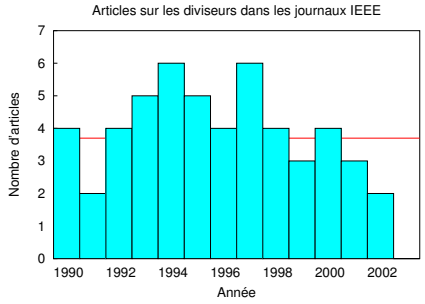
$$x_{i+1} = x_i - \frac{{x_i}^2 - c}{2x_i} = x_i - \frac{x_i}{2} + \frac{c}{2x_i} = \frac{1}{2}\left(x_i + \frac{c}{x_i}\right)$$

Seconde idée : utiliser Newton avec la fonction $f(x) = \frac{1}{x^2} - c$ qui a pour solution $\frac{1}{\sqrt{c}}$ (ensuite on multiplie par c pour avoir $\sqrt{c}$).

$$x_{i+1} = x_i - \frac{\frac{1}{x^2} - c}{-\frac{2}{{x_i}^3}} = x_i + \frac{x_i - c x_i^3}{2} = \frac{x_i}{2}\left(3 - c x_i^2\right)$$

Chaque itération fait intervenir 3 multiplications et une addition.

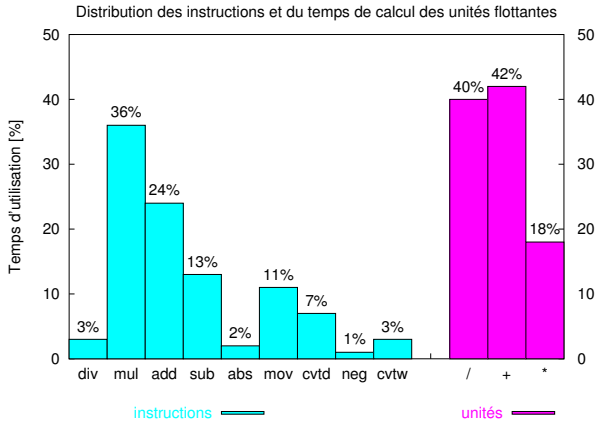
La recherche sur les diviseurs



- SRT base 4 dans le Pentium ou SRT base 8 dans l'Ultra (3 étages de base 2 par itération).
- Newton dans Itanium, Pentium 4 (table pour initialisation + routine logicielle).

La division, une opération peu utilisée ? Oui, mais. . .

Rapport technique[7] de S. Oberman et M. Flynn : "*Design issues in floating-point division*", CSL-TR-94-647 Stanford University.



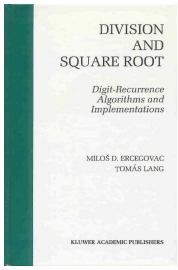
[7] SPECfp92 sur DECstation avec un MIPS R3000 (latences : 2c add, 5c mul, 19c div), compil. O3.

Pour en savoir plus. . .

Division and Square Root : Digit-Recurrence Algorithms and Implementations

Milos Ercegovac et Tomas Lang

1994
Kluwer
ISBN : 0–7923–9438–0




Division Algorithms and Implementations

Division Algorithms and Implementations

Stuart Oberman et Micheal Flynn

1997
IEEE Transactions on Computers
Vol. 46, No. 8, pp 833–854

Fin

Questions ?

Pour me contacter :

- arnaud.tisserand@ens-lyon.fr
- http://perso.ens-lyon.fr/arnaud.tisserand/
- Laboratoire LIP. ENS Lyon. 46 allée d'Italie. F-69364 Lyon cedex 07.

Merci.

Annexes

Représentation des entiers relatifs

Il existe différentes représentations possibles pour les entiers signés :

- signe et magnitude (valeur absolue)

$$A = (s_a a_{n-2} \dots a_1 a_0) = (-1)^{s_a} \times \sum_{i=0}^{n-2} a_i 2^i$$

- complément à la deux

$$A = (a_{n-1} a_{n-2} \dots a_1 a_0) = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

- biaisée (souvent $B = 2^{n-1} - 1$)

$$A = A_{math} + B$$

- ...

Représentation des entiers relatifs (suite)

entier	représentations		
	signe/magnitude	complément 2	biaisée (B=7)
–8	—	1000	—
–7	1111	1001	0000
–6	1110	1010	0001
–5	1101	1011	0010
–4	1100	1100	0011
–3	1011	1101	0100
–2	1010	1110	0101
–1	1001	1111	0110
0	0000	0000	0111
1	0001	0001	1000
2	0010	0010	1001
3	0011	0011	1010
4	0100	0100	1011
5	0101	0101	1100
6	0110	0110	1101
7	0111	0111	1110
8	—	—	1111

Trace de l'exécution du programme d'intégration

```
      10 8.410365e-02
      20 3.964663e-02
      50 1.533222e-02
     100 7.583203e-03
     200 3.769578e-03
     500 1.501383e-03
    1000 7.505436e-04
    2000 3.938694e-04
    5000 1.571794e-04
   10000 6.193113e-05
   20000 2.453346e-04
   50000 2.440791e-04
  100000 2.556424e-04
  200000 2.688150e-04
  500000 2.406953e-03
 1000000 9.144427e-03
 2000000 1.050532e-02
 5000000 3.374992e-03
10000000 2.150589e-02
20000000 3.068528e-01
50000000 1.931472e-01
```

Trace de l'exécution du programme de Gentleman

```
a = 1.000000    b = 1.000000
--- boucle1 ----------------------------------------------
64 itération(s)

                    a = 18446744073709551616.000000
              a + 1.0 = 18446744073709551616.000000
        (a + 1.0) - a = 0.000000
((a + 1.0) - a) - 1.0 = -1.000000

--- boucle2 ----------------------------------------------
                b = 1.000000
            a + b = 18446744073709551616.000000
      (a + b) - a = 0.000000
((a + b) - a) - b = -1.000000
1 itération(s)

                b = 2.000000
            a + b = 18446744073709551616.000000
      (a + b) - a = 2.000000
((a + b) - a) - b = 0.000000

Gentleman : 2.000000
```

Arithmétique exacte

Paul Zimmerman

Projet Spaces
LORIA/INRIA Lorraine

« Ce que j'aimerais dans ce cours, ce sont plus les algorithmes qui permettent d'effectuer des opérations exactes que leur utilisation. »

Exact vs Inexact

Exemple : calcul de $\sqrt{x}$ pour $x = 474256143563$ (12 chiffres).

Méthode « inexacte » :

```
> x:=474256143563:
> Digits:=20: evalf(sqrt(x));

                          688662.57598551120919
```

Méthode « exacte » :

```
> X:=1012*x: s:=isqrt(X): r:=X-s2: s, r;

                       688662575986, -673223872196
```

Avantages du calcul exact

Principe de base : calculer aussi le terme d'erreur (reste) en se ramenant à des calculs entiers.

• le calcul d'arrondi devient facile (ici il suffit de regarder le signe de r pour un arrondi dirigé, ou de comparer $|r|$ à $|s \pm \frac{1}{4}|$ pour l'arrondi au plus proche).

• le calcul lui-même peut être accéléré (typiquement l'itération de Newton commence par re-calculer le terme d'erreur).

• le calcul d'erreur est facilité par les instructions du type fma (*fused multiply add*).

Calcul exact des corrections (1)

Boldo, Daumas, *Representing correcting terms for possibly underflowing floating-point operations*, Arith'16, 2003.

Addition : erreur (arrondi au plus proche) représentable sur n bits.

Algorithme TwoSum (Knuth 1973).
$z \leftarrow x \oplus y$
$y_v \leftarrow z \ominus x$
$x_v \leftarrow z \ominus y_v$
$y_r \leftarrow y \ominus y_v$
$x_r \leftarrow x \ominus x_v$
$\epsilon \leftarrow x_r \oplus y_r$

Prouvé par Shewchuk en base 2 (précision ≥ 3), Priest (avec condition $|a \oplus a| \leq 2|a|$), Knuth (*"very intricate"*).

Calcul exact des corrections (2)

Multiplication : facile avec fma.

$z \leftarrow x \otimes y$
$\epsilon \leftarrow \circ(xy - z)$.

Division : pas toujours possible sous forme $x/y = z + \epsilon$, mais ok avec $x = yz + \epsilon$ (cf. multiplication).

Idem pour racine carrée : $x = s2 + \epsilon$.

Règle 1 : Tous les coups sont permis

Deux grandes classes d'algorithmes pour les calculs sur les entiers :

- classiques (par les poids forts)
- p-adiques (par les poids faibles)

Avantage des algos p-adiques : tout exact (les retenues vont dans le bon sens).

Exemple : division exacte $q = \frac{a}{b} : q = \frac{a}{b} \bmod 2^n$ pour $n \geq l(a) - l(b) + 1$ (Jebelean, 1993).

Krandick, Jebelean, 1996 : on gagne encore un facteur 2 en opérant par les deux bouts !

Règle 2 : Diviser pour régner

Théorème. Tout algorithme a une variante sous-quadratique.

PREUVE. Ajuster la taille du mot de base.

Exemple : la division (Burnikel, Ziegler, 1998).

Pour diviser $2n$ mots par n mots, il suffit de savoir diviser n mots par $n/2$ mots :

RecursiveDivide $(a : 2n,\ b : n)$
$(q_1, r_1) \leftarrow$ RecursiveDivide$(a_1 : n, b_1 : n/2)$
$a' \leftarrow r_1\beta^n + a_0 : 3n/2$
$(q_0, r_0) \leftarrow$ RecursiveDivide$(a'_1 : n, b_1 : n/2)$
$q \leftarrow q_1\beta^{n/2} + q_0$
$r \leftarrow r_0\beta^{n/2} + a'_0 : n$
Adjust (q, r) if necessary.

Règle 3 : Ne pas calculer des valeurs connues

Itération de Newton pour le calcul de $\sqrt{a}$:

$$x_{k+1} = x_k + \frac{1}{2x_k}(a - x_k 2).$$

Supposons que x_k a n bits (corrects). On sait donc que les n bits de poids fort de $x_k 2$ annulent ceux de a : il suffit donc de calculer les n bits de poids faible (produit court).

Règle 4 : Ne pas calculer des valeurs inutiles

Itération de Newton pour le calcul de $1/a$:

$$x_{k+1} = x_k + x_k(1 - ax_k).$$

Supposons que x_k a n bits corrects. Il faut prendre $2n$ bits de a pour obtenir $2n$ bits corrects dans x_{k+1}. Le produit ax_k fait donc $3n$ bits : les n bits de poids s'annulent avec 1, les n bits de poids faible sont donc inutiles.

Hanrot, Quercia, Z., *The Middle Product Algorithm, I*, AAECC, 2004.

Règle 5 : Réutiliser ce qui est connu

Exemple : *The* $x^n - 1$ *trick* (Berstein, 1999).

Les algorithmes style FFT calculent souvent une convolution cyclique (mod $2^n - 1$) ou négacyclique (mod $2^n + 1$).

On veut calculer un produit $n \times n$ dont une moitié est déjà connue (produit court) : $xy = z_0 + z_1$.

Au lieu de calculer xy sur $2n$ bits, on calcule xy mod $2^n - 1 = z_0 + z_1$. Une simple soustraction donne la moitié cherchée.

Règle 6 : Toujours faire simple

Se ramener autant que possible à des multiplications.

Exemple : pour calculer une racine carrée, on peut soit utiliser l'itération directe de Newton :

$$x_{k+1} = x_k + \frac{1}{2}(x_k + \frac{a}{x_k}),$$

ou bien calculer la racine carrée inverse via :

$$x_{k+1} = x_k + \frac{x_k}{2}(1 - ax_k 2).$$

Si la division est coûteuse par rapport à la multiplication, on préfèrera la seconde méthode.

Règle 7 : Se ramener à ce qui est facile à calculer

Si on sait calculer un inverse en $cM(n)$, alors on sait calculer un quotient en $(c+1)M(n)$.

Si on sait calculer une racine carrée inverse en $dM(n)$, alors on sait calculer une racine carrée en $(d+1)M(n)$.

On sait même faire mieux : $(c+1/2)M(n)$ et $(d-1/2)M(n)$.

Idée : incorporer le dividende lors de la dernière itération de Newton.

Karp, Markstein, *High-Precision Division and Square Root*, ACM TOMS, 1997.

Règle 8 : Changer de domaine au besoin

Exemple : multiplication de Montgomery pour les calculs mod n.

Si $n < \beta^l$, on remplace a par $\hat{a} = a\beta^l$.

$$a \pm b \to \hat{a} \pm \hat{b}, \qquad ab \to REDC(\hat{a}, \hat{b}) = \frac{\hat{a}\hat{b}}{\beta^l}$$

Comment calculer $\hat{a}\hat{b}/\beta^l$?

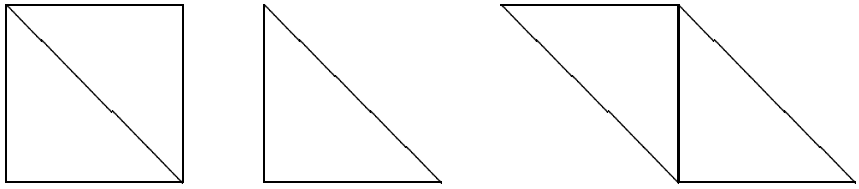
0. précalculer $\mu = -1/n \bmod \beta$

1. $c \leftarrow \hat{a}\hat{b}$

2. faire l fois : $\lambda = \mu c \bmod \beta$, $c = (c + \lambda n)/\beta$ [division exacte]

P. Montgomery, *Modular Multiplication Without Trial Division*, Math. of Comp., 1985.

Règle 9 : Faire un dessin

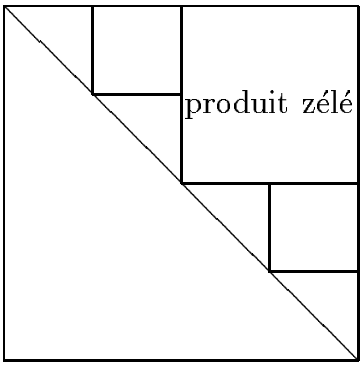


Multiplication Produit court Produit médian

Multiplication, produit médian $\leq$ 2 produits courts

Règle 10 : Anticiper

Produit « détendu » :

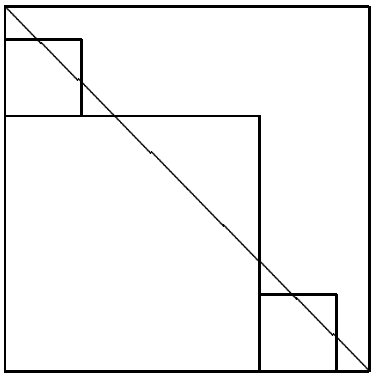


J. van der Hoeven, *Relax, but don't be too lazy*, JSC, 2002.

Règle 11 : Calculer plus au besoin

[exception à la règle 4 : Ne pas calculer des valeurs inutiles]

Exemple : calcul d'un produit court.



Mulders, *On short multiplications and divisions*, AAECC, 2000.

Pour en savoir plus

R. P. Brent, P. Zimmermann, *Modern Computer Arithmetic*, en préparation.

P. Zimmermann, *Arithmétique en précision arbitraire*, Réseaux et Systèmes Répartis, Calculateurs Parallèles, 2001.

Introduction à la Théorie des Systèmes Hybrides

Responsable : Jean-Guillaume Dumas, UJF.

Conférenciers : Thao Dang, Antoine Girard.

Sommaire

Introduction à la Théorie des Systèmes Hybrides

Thao Dang et Antoine Girard

VERIMAG
2 avenue de Vignate,
38610 Gières, France
LMC-IMAG
51 rue des Mathématiques,
38041 Grenoble, France
{Thao.Dang,Antoine.Girard}@imag.fr

1 Définition et Exemples

1.1 Définition d'un système hybride

Tout système impliquant des processus continus et des phénomènes discrets peut être vu comme un système hybride. Par extension, lorsque dans un même système physique certaines grandeurs varient très rapidement (quasi-instantanément) par rapport aux autres alors une modélisation hybride de ce système est envisageable et donne souvent de bien meilleurs résultats qu'une modélisation continue.
Ainsi, les domaines d'application des systèmes hybrides sont extrêmement nombreux et variés; on peut citer entre autres l'informatique [19], l'industrie automobile [6], la robotique [4], le contrôle du trafic aérien [23] et la biologie [5].
Par conséquent, une définition unifiée des systèmes hybrides pouvant servir d'environnement théorique à la description de ces phénomènes est très difficile. La définition suivante nous semble relativement générale. Elle est notamment motivée par [7, 22].

Définition 1.1 (Système Hybride) *Un système hybride est un septuple*

$$\mathcal{H} = (\mathcal{Q}, \mathcal{E}, \mathcal{D}, \mathcal{U}, \mathcal{F}, \mathcal{G}, \mathcal{R})$$

où :

1. $\mathcal{Q}$ *est l'ensemble dénombrable des* états discrets *(ou modes).*
2. $\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{Q}$ *est l'ensemble des* arêtes *(ou transitions).*
3. $\mathcal{D} = \{D_q, q \in \mathcal{Q}\}$ *est la collection des* domaines.
   $\forall q \in \mathcal{Q}$, D_q *est un sous-ensemble de* $\mathbb{R}^n$ *d'intérieur non-vide.*
4. $\mathcal{U} = \{U_q, q \in \mathcal{Q}\}$ *est la collection des* domaines de contrôle.
   $\forall q \in \mathcal{Q}$, U_q *est un sous-ensemble de* $\mathbb{R}^p$.
5. $\mathcal{F} = \{f_q, q \in \mathcal{Q}\}$ *est la collection de* champs de vecteurs.
   $\forall q \in \mathcal{Q}$, $f_q : D_q \times U_q \to \mathbb{R}^n$.

6. $\mathcal{G} = \{G_e, e \in \mathcal{E}\}$ *est la collection des* gardes.
   $\forall e = (q, q') \in \mathcal{E}$, $G_e \subseteq D_q$.
7. $\mathcal{R} = \{R_e, e \in \mathcal{E}\}$ *est la collection des fonctions* resets.
   $\forall e = (q, q') \in \mathcal{E}$, $R_e : G_e \to 2^{D_{q'}}$ *où* $2^{D_{q'}}$ *dénote l'ensemble des parties de* D_q.
   On suppose que pour tout $x \in G_e$, $R_e(x) \neq \emptyset$.

Remarque 1.1 *Il arrive que les équations différentielles associées aux éléments de* $\mathcal{F}$ *soient autonomes (i.e.* $f_q : D_q \to \mathbb{R}^n$*). Dans ce cas, la donnée des domaines de contrôle est superflue et le système hybride est alors défini par le sextuple* $(\mathcal{Q}, \mathcal{E}, \mathcal{D}, \mathcal{F}, \mathcal{G}, \mathcal{R})$.

Intuitivement, l'état du système hybride peut être décrite par deux variables : la première discrète, notée $q(t)$ à valeur dans $\mathcal{Q}$, la deuxième continue, notée $x(t)$ à valeur dans $\mathbb{R}^n$.
Un exemple de fonctionnement (on parle d'exécution) est montré sur la figure 1. A l'instant initial ($t = t_0$), $q(t_0) = q_0$ et la valeur initiale de la variable continue $x(t_0)$ est un élément du domaine D_{q_0}. Cette dernière évolue alors en suivant l'équation différentielle $x'(t) = f_{q_0}(x(t), u(t))$, où la fonction u prend ses valeurs dans le domaine de contrôle U_{q_0}. Enfin, lorsque la trajectoire $x(t)$ atteint une garde G_e à l'instant t_1 (avec ici $e = (q_0, q_1)$), la variable discrète $q(t)$ peut alors prendre la valeur q_1. La variable continue est réinitialisée à une valeur de l'ensemble $R_e(x(t_1^-)) \subseteq D_{q_1}$. On répète alors le même processus avec une nouvelle équation différentielle et de nouvelles gardes.
Par la suite, nous définirons rigoureusement la notion d'exécution d'un système hybride.

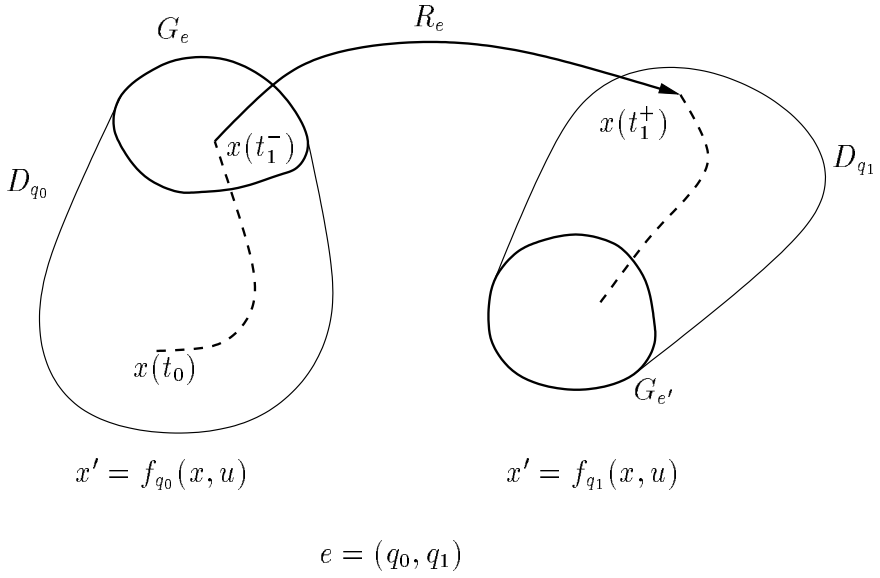


FIG. 1: Exemple d'exécution d'un système hybride

Un système hybride est donc constitué de deux composantes complémentaires couplées.
La première, donnée par le couple $(\mathcal{Q}, \mathcal{E})$, détermine la dynamique discrète du système. Elle est décrite par l'automate dont les sommets sont les éléments de $\mathcal{Q}$ et les arêtes ceux de $\mathcal{E}$.

La deuxième composante du système hybride est décrite par le triplet $(\mathcal{D}, \mathcal{U}, \mathcal{F})$ et décrit la dynamique continue du système. Elle consiste en une collection de systèmes dynamiques indexés sur les éléments de $\mathcal{Q}$. Ainsi, la valeur de la variable discrète $q(t)$ détermine-t-elle l'équation différentielle gouvernant l'évolution de la variable continue $x(t)$. A chaque instant t, tel que $q(t) = q$,

$$\begin{cases} x'(t) = f_q(x(t), u(t)), \\ (x(t), u(t)) \in D_q \times U_q. \end{cases}$$

Le couplage des deux composantes se fait par l'introduction de l'ensemble des gardes $\mathcal{G}$. En effet, pour qu'une transition de l'automate $e \in \mathcal{E}$ puisse se faire à un instant t, il est nécessaire que $x(t)$ appartienne à l'ensemble G_e.
Ainsi la composante discrète du système contrôle la composante continue qui, rétroactivement, détermine, à chaque instant, l'ensemble (qui peut être vide) des transitions possibles de l'automate. L'ensemble résultant constitu ce que l'on appelle automate hybride.

1.2 Exemples de systèmes hybrides

Le formalisme des systèmes hybrides est très général et englobe de nombreuses classes de modèles de la théorie du contrôle. Dans ce paragraphe, nous présentons certaines d'entre elles qui nous semblent significatives.

1.2.1 Systèmes dynamiques impulsionnels

Un système dynamique impulsionnel [21] décrit l'évolution d'une variable continue $x(t)$ régie par une équation différentielle sous contrainte. Lorsque $x(t)$ vérifie certaines conditions, il est alors possible de lui donner une *impulsion*, c'est à dire de lui affecter une nouvelle valeur.
Dans le cadre des systèmes hybrides, les systèmes dynamiques impulsionnels correspondent aux systèmes possédant un seul mode ($\mathcal{Q} = \{q\}$) et une transition $\mathcal{E} = \{e = (q, q)\}$ autorisant la réinitialisation de la variable continue via la fonction reset R_e.

Exemple 1.1 (Balle rebondissante [17]) *On considère une balle de masse m soumise à l'action de la gravité. On la laisse tomber d'une altitude z_0 avec une vitesse initiale nulle. L'altitude $z(t)$ de la balle suit donc l'équation différentielle issue de la mécanique classique $mz''(t) = -mg$. Quand $z(t) = 0$, la balle touche le sol et rebondit en perdant une fraction de son énérgie :*

$$z'(t^+) = -cz'(t^-), \text{ avec } c \leq 1.$$

En posant $x_1(t) = z(t)$, $x_2(t) = z'(t)$ et en utilisant le formalisme de la définition 1.1, le modéle hybride de la balle rebondissante est donné par :

1. $\mathcal{Q} = \{q\}$
2. $\mathcal{E} = \{e = (q, q)\}$
3. $D_q = \mathbb{R}^+ \times \mathbb{R}$
4. $f_q(x_1, x_2) = (x_2, -g)$
5. $G_e = \{x_1 = 0\}$
6. $R_e(x_1, x_2) = \{(x_1, -cx_2)\}$

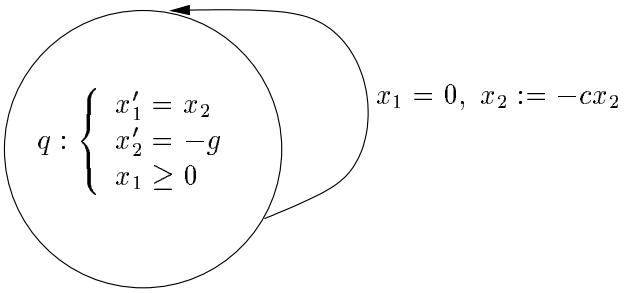


FIG. 2: Automate hybride de la balle rebondissante

1.2.2 Systèmes dynamiques par morceaux

Les équations différentielles à second membre discontinu [13] ou défini par morceaux interviennent dans de nombreux problèmes d'ingénierie; notamment en électronique où les modèles de composants sont souvent linéaires par morceaux [8].

Le domaine de définition D de l'équation différentielle est un sous-ensemble fermé et connexe de $\mathbb{R}^n$. D est découpé en sous-domaines $\{D_q, q \in \mathcal{Q}\}$, fermés, d'intérieur non-vide et deux à deux disjoints, et tels que :

$$\bigcup_{q \in \mathcal{Q}} D_q = D.$$

Sur chaque domaine D_q, on définit un champ de vecteur f_q.
La trajectoire $x(t)$ du système dynamique par morceaux se construit de la manière suivante (voir figure 3). Si $x(t_0)$ appartient à l'intérieur du domaine D_{q_0}, alors $x(t)$ est solution de l'équation différentielle associée au champ de vecteur f_{q_0} jusqu'à l'instant t_1 où $x(t)$ atteint la frontière séparant le domaine D_{q_0} du domaine D_{q_1}. $x(t)$ devient alors solution de l'équation différentielle associée au champ de vecteur f_{q_1}.

Dans le cadre de la définition 1.1, un système dynamique par morceaux est donc un système hybride présentant les particularités suivantes.

– $\mathcal{Q}$ est l'ensemble indexant les sous domaines D_q.

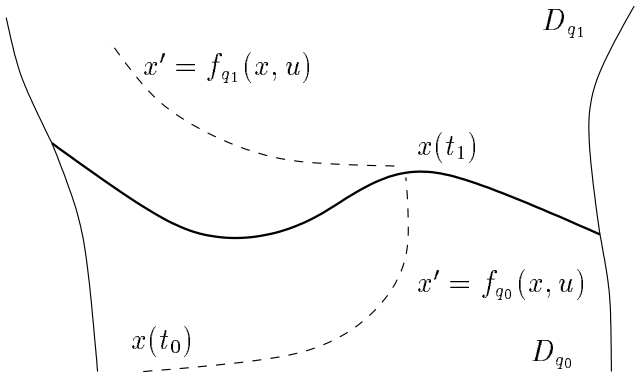


FIG. 3: Trajectoire du système dynamique par morceaux

- $\mathcal{E} = \{(q,q') \in \mathcal{Q} \times \mathcal{Q}, \partial D_q \cap \partial D_{q'} \neq \emptyset\}$.
  On ne peut passer du domaine D_q au domaine $D_{q'}$ que s'ils ont une frontière commune.
- $\mathcal{G} = \{G_e, e \in \mathcal{E}\}$. $\forall e = (q,q') \in \mathcal{E},\ G_e = \partial D_q \cap \partial D_{q'}$.
  On ne peut passer du domaine D_q au domaine $D_{q'}$ qu'en franchissant leur frontière commune.
- $\mathcal{R} = \{R_e, e \in \mathcal{E}\}$. $\forall e \in \mathcal{E},\ \forall x \in G_e\ , R_e(x) = \{x\}$.
  Il n'y a pas de réinitialisation de la variable continue.

Exemple 1.2 (Oscillateur électronique à valve [1]) *L'oscillateur éléctronique à valve est un circuit composé d'une valve électronique, d'un circuit oscillant de type RLC et d'un dispositif de rétroaction par induction électromagnétique (voir figure 4). Une attention particulière a été portée à ce montage car c'est le circuit le plus simple exhibant des oscillations.*

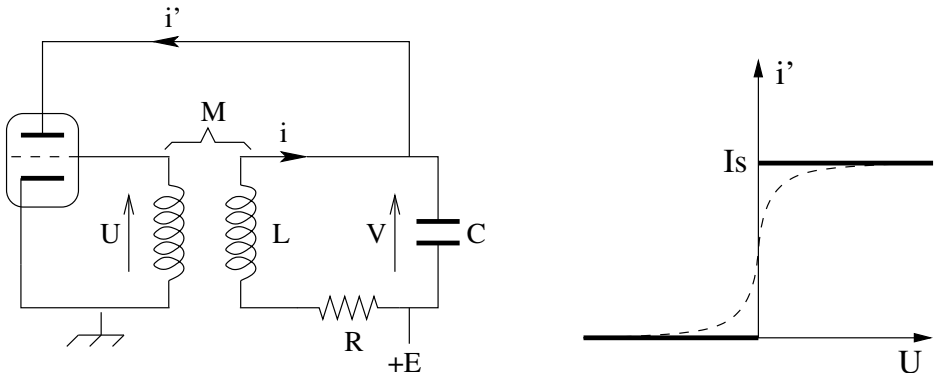


FIG. 4: Gauche : oscillateur éléctronique à valve. Droite : caractéristique de la valve.

En appliquant la loi des mailles dans le circuit RLC et la loi des noeuds au point

de branchement de la valve électronique et du circuit RLC, on obtient les équations

$$Ri + V + L\frac{di}{dt} = 0,\ i = i' + C\frac{dV}{dt} \implies LC\frac{d^2i}{dt^2} + RC\frac{di}{dt} + i = i'.$$

Le courant entrant à l'anode de la valve est donné par l'intensité i' et dépend exclusivement de la tension U appliquée à la grille. La caractéristique de la valve $i' = i'(U)$ est représentée en pointillé sur la figure 4.
Si les oscillations de la tension U sont d'amplitude suffisante, l'intensité i' est, la plupart du temps, égale soit à 0 soit à I_s. On peut alors, raisonnablement, représenter les propriétés d'une telle valve par la caractéristique idéale dessinée en gras sur la figure. Par conséquent,

$$LC\frac{d^2i}{dt^2} + RC\frac{di}{dt} + i = \begin{cases} 0 & si\ U < 0 \\ I_s & si\ U > 0. \end{cases}$$

La tension U est la différence de potentiels aux bornes de la bobine couplée à la bobine du circuit RLC. Le courant d'intensité i crée un champ magnétique à l'intérieur de la bobine du circuit RLC. Ce champ magnétique, à son tour, induit un courant dans la bobine reliée à la grille de la valve. Cette action rétroactive se traduit par l'équation

$$U = -M\frac{di}{dt}.$$

On suppose que les bobines sont agencées de manière à avoir $M < 0$. On a donc

$$LC\frac{d^2i}{dt^2} + RC\frac{di}{dt} + i = \begin{cases} 0 & si\ \frac{di}{dt} < 0 \\ I_s & si\ \frac{di}{dt} > 0. \end{cases}$$

En posant, $x_1 = i$, $x_2 = \frac{di}{dt}$, l'oscillateur électronique à valve peut être modélisé par le système dynamique par morceaux :

$$x_1'(t) = x_2(t),\ x_2'(t) = \begin{cases} -\frac{1}{LC}x_1(t) - \frac{R}{L}x_2(t) & si\ x_2(t) < 0 \\ -\frac{1}{LC}x_1(t) - \frac{R}{L}x_2(t) + \frac{I_s}{LC} & si\ x_2(t) > 0. \end{cases}$$

L'automate hybride correspondant est représenté sur la figure 5.

1.2.3 Systèmes dynamiques à commutation

Un système dynamique à commutation ou *switched system* (voir par exemple [12]) est un système hybride où la variable discrète $q(t)$ n'est pas vue comme une variable d'état mais comme une variable de contrôle. Ainsi, l'évolution de $q(t)$ n'est pas contrainte par un système de gardes mais donnée par un individu extérieur.

Par conséquent, d'après la définition 1.1, les systèmes dynamiques à commutation vérifient la propriété suivante :

$$\forall e = (q, q') \in \mathcal{E},\ G_e = D_q.$$

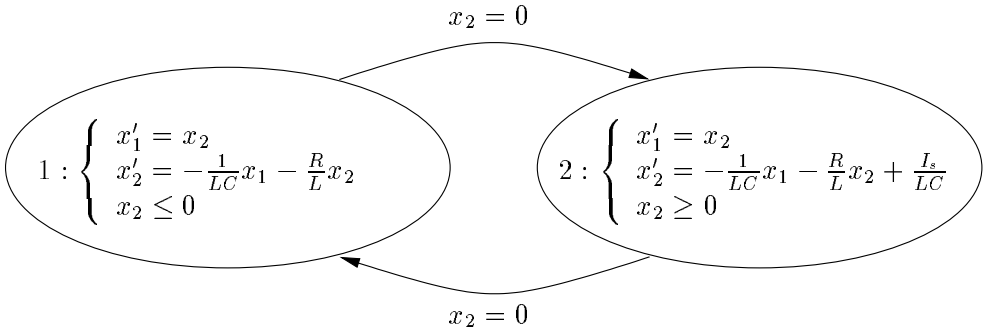


FIG. 5: Automate hybride de l'oscillateur électrique à valve

Exemple 1.3 (boîte de vitesse [14]) *Nous présentons ici un modèle simplifié d'un véhicule motorisé à deux vitesses se déplaçant sur un axe. On note $x_1(t)$ la position du véhicule sur l'axe et $x_2(t)$ sa vitesse.*
Le conducteur agit sur le système par deux moyens. D'abord, via la pédale d'accélération, il contrôle l'apport de carburant représenté par la variable $u(t) \in [0, u_{max}]$. Ensuite, le véhicule est équipé d'une boîte de vitesse à deux rapports, on note $q(t) \in \{1, 2\}$ le rapport actif.
Le système est décrit par les équations

$$x_1'(t) = x_2(t),\ x_2'(t) = u(t)g_{q(t)}(x_2(t)) - kx_2(t).$$

Les deux rapports de la boîte de vitesse sont caractérisés par les fonctions g_1 et g_2 (figure 6). Elles représentent le rendement du moteur et et relient l'accélération du véhicule, sa vitesse, l'apport de carburant et le rapport actif.

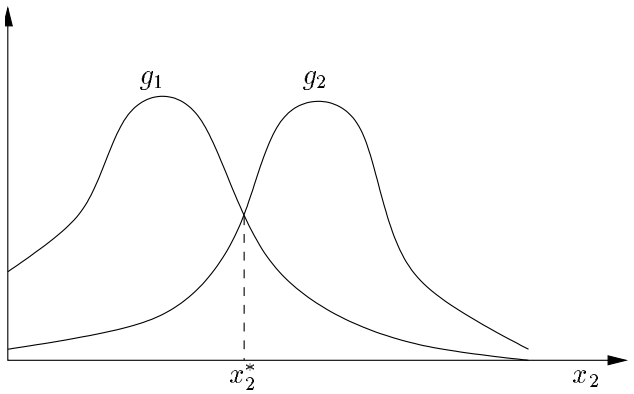


FIG. 6: Caractéristique de la boîte de vitesse

Le véhicule peut donc être modélisé par le système hybride suivant :

1. $\mathcal{Q} = \{1, 2\}$.
2. $\mathcal{E} = \{(1, 2), (2, 1)\}$.

3. $D_1 = D_2 = \mathbb{R}^2$.
4. $U_1 = U_2 = [0, u_{max}]$.
5. $f_q(x_1, x_2, u) = (x_2, ug_q(x_2) - kx_2)$, $q \in \mathcal{Q}$.
6. $G_{(1,2)} = G_{(2,1)} = \mathbb{R}^2$.
7. $R_{(1,2)}(x) = R_{(2,1)}(x) = \{x\}$.

1.2.4 Des systèmes à commutation aux systèmes hybrides

Un problème classique lié aux systèmes dynamiques à commutation consiste à déterminer une loi de rétroaction de la variable $x(t)$ sur le contrôle discret $q(t)$. Les changements de valeur de $q(t)$ ne sont plus décidés par l'extérieur mais déterminés par $x(t)$.
Dans le formalisme des systèmes hybrides, cela revient à spécifier les gardes du système qui détermineront les conditions de changement de la variable $q(t)$.

Exemple 1.4 (boîte de vitesse automatique) *Reprenons l'exemple 1.3, on veut équiper notre véhicule d'une boîte de vitesse automatique. Pour celà, on équipe le véhicule d'un ordinateur de bord.*
Celui ci reçoit est informé en temps réel sur la valeur de la vitesse du véhicule et détermine, en conséquence, le rapport le mieux adapté (voir figure 7).

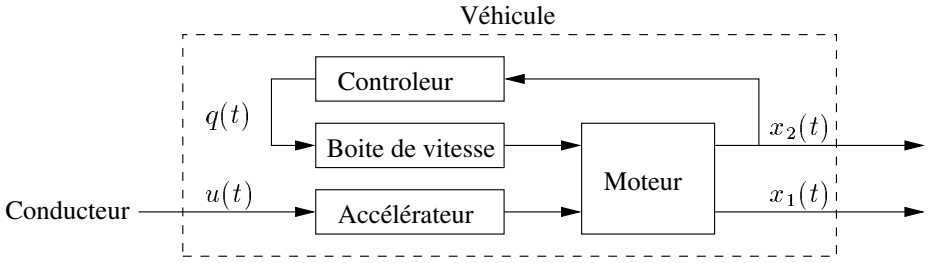


FIG. 7: Schéma du véhicule équipé d'une boîte de vitesse automatique

La stratégie la plus simple et la plus facilement implémentable consiste à sélectionner le rapport le plus efficace; maximisant l'accélération du véhicule pour des apports de carburant égaux.
Si à un instant t,

$$u(t)g_1(x_2(t)) - kx_2(t) \geq u(t)g_2(x_2(t)) - kx_2(t),$$

ou de manière équivalente,

$$g_1(x_2(t)) \geq g_2(x_2(t)),$$

alors le premier rapport est plus approprié. Donc, d'après la caractéristique de la boîte de vitesse (figure 6), si le véhicule à une allure inférieure à x_2^*, il vaut mieux

choisir le premier rapport.

La figure 8 présente l'automate hybride modélisant le véhicule équipé d'un contrôleur implémentant cette stratégie.

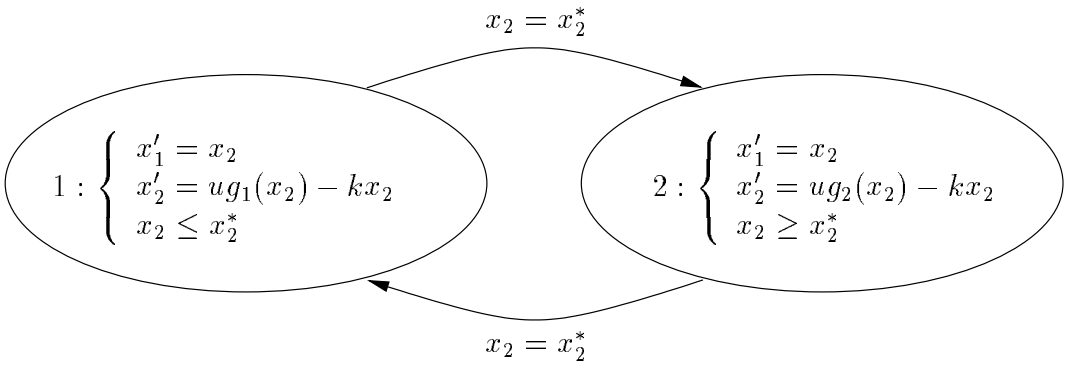


FIG. 8: Automate hybride de la boîte de vitesse automatique

2 Notion de Solution d'un Système Hybride

Dans le chapitre précédent, nous avons abordé, de manière intuitive, la notion de solution d'un système hybride. Dans cette partie, nous allons la formaliser. Les résultats exposés dans ce chapitre sont majoritairement inspirés par le travail de K.H. Johansson, J. Lygeros, S. Sastry et M. Egerstedt [17, 18].

2.1 Exécution d'un système hybride

2.1.1 Trajectoire temporisée

Nous avons vu que l'exécution d'un système hybride était caractérisée par l'évolution des variables $q(t)$ et $x(t)$. La variable discrète $q(t)$ est constante par morceaux et est donc entièrement donnée par les séquences $\{t_i\}_{i=0}^{i=N}$ des points de discontinuité et des valeurs successives de $q(t)$. La notion de trajectoire temporisée permet de décrire la séquence des instants de transition du système.

Définition 2.1 (Trajectoire temporisée) *Une trajectoire temporisée $\tau = \{I_i\}_{i=0}^{i=N}$ est une séquence finie ou infinie d'intervalles de $\mathbb{R}$ vérifiant*

1. *Pour $i < N$, $I_i = [t_i, t_{i+1}]$ avec $t_i \leq t_{i+1}$;*
2. *Si N est fini, $I_N = [t_N, t_{N+1}]$ avec $t_N \leq t_{N+1}$, ou $I_N = [t_N, t_{N+1}[$ avec $t_N < t_{N+1}$ (on peut avoir $t_{N+1} = +\infty$).*

On notera, $t \in \tau$, si il existe $i \in \{0, \ldots, N\}$, tel que $t \in I_i$.
On appelle longueur de la trajectoire temporisée la quantité $\sum_{i=0}^{i=N}(t_{i+1} - t_i)$. Une trajectoire temporisée de longueur finie (respectivement infinie) est dite limitée (respectivement illimitée).
Les trajectoires temporisées peuvent être séparées en quatre classes principales. En effet, soit τ une trajectoire temporisée, elle peut être :

- finie, limitée, $N < +\infty$, $t_{N+1} < +\infty$;
- finie, illimitée, $N < +\infty$, $t_{N+1} = +\infty$;
- infinie, limitée, $N = +\infty$, $\lim_{i \to +\infty} t_i < +\infty$;
- infinie, illimitée, $N = +\infty$, $\lim_{i \to +\infty} t_i = +\infty$.

Sur l'ensemble des trajectoires temporisées, on définit une relation d'ordre partielle.

Définition 2.2 (Préfixe) *Soit $\tau = \{I_i\}_{i=0}^{i=N}$, $\tau' = \{J_i\}_{i=0}^{i=M}$ deux trajectoires temporisées. On dit que τ est un préfixe de τ' (noté $\tau \leq \tau'$) si elles sont identiques ou si τ est finie et*

$$M \geq N,\ I_i = J_i,\ \textit{pour } i \in \{0, \ldots N-1\},\ \textit{et } I_N \subseteq J_N.$$

Si de plus $\tau \neq \tau'$, on dit que τ est un préfixe strict de τ' (noté $\tau < \tau'$).

Il est clair que si τ est un préfixe strict d'une trajectoire temporisée, alors, nécessairement, τ est finie et limitée.

2.1.2 Trajectoire hybride

Ainsi, une trajectoire temporisée τ représente la séquence des intervalles sur lesquelles les variables du système hybride $q(t)$ et $x(t)$ évoluent continûment.
Dans le premier chapitre, on a vu qu'une transition pouvait se produire à un instant t, lorsque $x(t)$ appartient à une garde. La transition se produit alors en temps nul; on affecte une nouvelle valeur à $q(t)$ et $x(t)$.
On voit donc qu'à l'instant t où la transition s'effectue, les valeurs de $q(t)$ et $x(t)$ ne sont pas définies de manière unique. Pour lever l'ambiguité, on définit la notion de trajectoire hybride (voir figure 9).

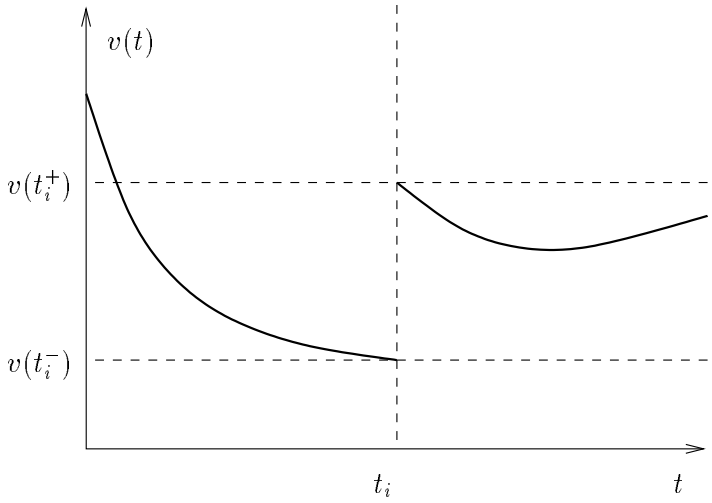


FIG. 9: Exemple d'une trajectoire hybride (τ, v)

Définition 2.3 (Trajectoire hybride) *Une trajectoire hybride dans un ensemble V est un doublet (τ, v) tel que :*

1. *$\tau = \{I_i\}_{i=0}^{i=N}$ est une trajectoire temporisée;*
2. *v est une séquence d'applications continues $\{v_i\}_{i=0}^{i=N}$, avec*

$$\forall i \in \{0, \ldots, N\}, v_i : I_i \to V \ .$$

Les bornes des intervalles de la trajectoire temporisée τ représentent les instants auxquels les transitions se produisent.
On note $v : \tau \to V$. Pour tout $t \in \tau$, tel que $t \notin \{t_1, \ldots, t_N\}$, on définit

$$v(t) = v_i(t), \text{ si } t \in I_i \ .$$

Pour tout $t \in \{t_1, \ldots, t_N\}$, $v(t)$ prend deux valeurs que l'on notera

$$v(t_i^-) = v_{i-1}(t_i), \ v(t_i^+) = v_i(t_i) \ .$$

On définit sur l'ensemble des trajectoires hybrides une relation d'ordre partiel.

Définition 2.4 (Préfixe) *Soient (τ, v) et (τ', v') deux trajectoires hybrides dans un ensemble V. On dit que (τ, v) est un préfixe de (τ', v') (noté $(\tau, v) \leq (\tau', v')$) si*

$$\tau \leq \tau' \text{ et, pour tout } t \in \tau,\ v(t) = v'(t).$$

Si de plus, $(\tau, v) \neq (\tau', v')$, alors on dit que (τ, v) est un préfixe strict de (τ', v') (noté $(\tau, v) < (\tau', v')$).

Définition 2.5 (Plus grand commun préfixe) *Soient (τ, v) et (τ', v') deux trajectoires hybrides, telles que $(t_0, v(t_0))=(t'_0, v'(t_0))$. On appelle plus grand commun préfixe la trajectoire hybride $(\hat{\tau}, \hat{v})$ vérifiant :*

1. $(\hat{\tau}, \hat{v}) \leq (\tau, v)$ *et* $(\hat{\tau}, \hat{v}) \leq (\tau', v')$;
2. $\forall (\bar{\tau}, \bar{v}),\ (\bar{\tau}, \bar{v}) \leq (\tau, v)$ *et* $(\bar{\tau}, \bar{v}) \leq (\tau', v') \implies (\bar{\tau}, \bar{v}) \leq (\hat{\tau}, \hat{v})$.

Pour deux trajectoires hybrides le plus grand commun préfixe représente donc leur plus grande partie commune avant que les deux trajectoires ne se séparent.

Proposition 2.1 *Soient (τ, v) et (τ', v') deux trajectoires hybrides, soit $(\hat{\tau}, \hat{v})$ leur plus grand commun préfixe, alors :*

- *soit $\hat{\tau} = \tau$ ou $\hat{\tau} = \tau'$;*
- *soit $\hat{\tau} = \{\hat{I}_i\}_{i=0}^{i=\hat{N}}$, avec $\hat{N} < +\infty$ et $\hat{I}_{\hat{N}} = [\hat{t}_{\hat{N}}, \hat{t}_{\hat{N}+1}]$.*

Preuve : Supposons $(\tau, v) \leq (\tau', v')$, alors $(\hat{\tau}, \hat{v}) = (\tau, v)$. De même, si $(\tau', v') \leq (\tau, v)$ alors $(\hat{\tau}, \hat{v}) = (\tau', v')$.
Supposons, maintenant (τ, v) et (τ', v') ne sont pas comparables. Il est clair que $\hat{\tau} < \tau$ et $\hat{\tau} < \tau'$ et donc $\hat{\tau}$ est finie et limitée.
Donc $\hat{\tau} = \{\hat{I}_i\}_{i=0}^{i=\hat{N}}$, avec $\hat{N} < +\infty$. Si $\hat{I}_{\hat{N}} = [\hat{t}_{\hat{N}}, \hat{t}_{\hat{N}+1}[$, alors puisque $\hat{\tau}$ est un préfixe strict de τ et de τ', on a $\hat{I}_{\hat{N}} \subset I_{\hat{N}}$ et $\hat{I}_{\hat{N}} \subset I'_{\hat{N}}$.
Par conséquent, $\hat{t}_{\hat{N}+1}$ appartient aux intervalles $I_{\hat{N}}$ et $I'_{\hat{N}}$.
De plus,

$$\forall t \in \hat{I}_{\hat{N}},\ v_{\hat{N}}(t) = \hat{v}_{\hat{N}}(t) = v'_{\hat{N}}(t).$$

Par continuité de $v_{\hat{N}}$ sur $I_{\hat{N}}$ et $v'_{\hat{N}}$ sur $I'_{\hat{N}}$.

$$v_{\hat{N}}(\hat{t}_{\hat{N}+1}) = \lim_{t \to \hat{t}_{\hat{N}+1}} v_{\hat{N}}(t) = \lim_{t \to \hat{t}_{\hat{N}+1}} v'_{\hat{N}}(t) = v'_{\hat{N}}(\hat{t}_{\hat{N}+1}).$$

On peut donc définir $(\bar{\tau}, \bar{v})$ le préfixe de (τ, v) et (τ', v'), tel que le dernier intervalle de $\bar{\tau}$ est $[\hat{t}_{\hat{N}}, \hat{t}_{\hat{N}+1}]$. On a de manière immédiate $(\hat{\tau}, \hat{v}) < (\bar{\tau}, \bar{v})$; $(\hat{\tau}, \hat{v})$ n'est donc pas le plus grand commun préfixe de (τ, v) et (τ', v').

■

2.1.3 Définition d'une exécution d'un système hybride

Nous pouvons maintenant définir la notion de solution d'un système hybride (on parle d'exécution). L'ensemble des éxecutions d'un système hybride est une partie de l'ensemble des trajectoires hybrides dans l'ensemble $\mathcal{Q} \times \mathbb{R}^n$.

Définition 2.6 (Exécution) *Une exécution d'un système hybride $\mathcal{H}$ est une trajectoire hybride (τ, q, x), où $q : \tau \to \mathcal{Q}$ et $x : \tau \to \mathbb{R}^n$, vérifiant*

1. **évolution continue :** *pour tout $i \in \{0, \ldots, N\}$, tel que $t_i < t_{i+1}$:*
   - *$q(t)$ est constante sur l'intervalle I_i;*
   - *$x(t) \in D_{q(t)}$ sur l'intervalle I_i;*
   - *il existe une application $u : [t_i, t_{i+1}[\to U_{q(t)}$, telle que*

$$\forall t \in [t_i, t_{i+1}[,\ x'(t) = f_{q(t)}(x(t), u(t)) .$$

2. **évolution discrète :** *pour tout $i \in \{1, \ldots, N\}$:*
   - *$e = (q(t_i^-), q(t_i^+)) \in \mathcal{E}$;*
   - *$x(t_i^-) \in G_e$;*
   - *$x(t_i^+) \in R_e(x(t_i^-))$.*

On dit qu'un système hybride accepte une exécution. Le triplet $(t_0, q(t_0), x(t_0))$ est appelé condition initiale de l'exécution.

Remarque 2.1 *Pour toute condition initiale (t_0, q_0, x_0), x_0 est un élément de D_{q_0}.*

2.1.4 Classification des exécutions

Un système hybride peut accepter une grande variété d'exécutions. Une classification de ces exécutions se révèle utile.

Définition 2.7 (Exécution maximale) *Une exécution (τ, q, x) est maximale si elle n'est le préfixe d'aucune autre exécution de $\mathcal{H}$.*

Définition 2.8 (Exécution finie, infinie) *Une exécution (τ, q, x) est dite :*

- *finie, si τ est finie et limitée;*
- *infinie, si τ est soit infinie, soit illimitée.*

On a de manière assez immédiate le résultat suivant.

Proposition 2.2 *Une exécution infinie est maximale.*

Preuve : Soit (τ, q, x) une exécution infinie. τ est soit infinie, soit illimitée et ne peut donc pas être le préfixe d'une autre trajectoire temporisée. Par conséquent, l'exécution (τ, q, x) n'est le préfixe d'aucune autre trajectoire hybride et est donc maximale.

■

Exemple 2.1 *On considère le système hybride $\mathcal{H}$ définit par :*

1. $\mathcal{Q} = \{1, 2\}$
2. $\mathcal{E} = \{(1, 2), (2, 1)\}$
3. $D_1 = D_2 = [-3, 3]$
4. $f_1(x) = 1$, $f_2(x) = -1$

5. $G_{(1,2)} = [1,2]$, $G_{(2,1)} = [-1,-2]$
6. $R_{(1,2)} = R_{(2,1)} = \{x\}$

$\mathcal{H}$ accepte les trois exécutions suivantes (voir figure 10).
Première exécution *:* (τ_1, q_1, x_1) *avec*

- $\tau_1 = \{[2i, 2i+2]\}_{i=0}^{i=+\infty}$;
- *Pour tout* $t \in [2i, 2i+2]$, $q_1(t) = \begin{cases} 1, & si\ i\ est\ pair \\ 2, & si\ i\ est\ impair; \end{cases}$
- *Pour tout* $t \in [2i, 2i+2]$, $x_1(t) = \begin{cases} -1 \quad +(t-2i), & si\ i\ est\ pair \\ 1 \quad -(t-2i), & si\ i\ est\ impair. \end{cases}$

τ_1 est infinie et illimitée, par conséquent l'exécution (τ_1, q_1, x_1) est infinie et donc maximale.
Deuxième exécution *:* (τ_2, q_2, x_2) *avec*

- $\tau_2 = \{[0,2],[2,4]\}$;
- *Pour tout* $t \in \tau$, $q_2(t) = \begin{cases} 1, & si\ t \in [0,2] \\ 2, & si\ t \in [2,4]; \end{cases}$
- *Pour tout* $t \in \tau$, $x_2(t) = \begin{cases} -1 \quad +t, & si\ t \in [0,2] \\ 3 \quad -t, & si\ t \in [2,4]. \end{cases}$

τ_2 est finie et limitée, par conséquent l'exécution (τ_2, q_2, x_2) est finie. De plus, il est clair que $(\tau_2, q_2, x_2) < (\tau_1, q_1, x_1)$ et donc (τ_2, q_2, x_2) n'est pas maximale.
Troisième exécution *:* (τ_3, q_3, x_3) *avec*

- $\tau_3 = \{[0,2],[2,6]\}$;
- *Pour tout* $t \in \tau$, $q_3(t) = \begin{cases} 1, & si\ t \in [0,2] \\ 2, & si\ t \in [2,6]; \end{cases}$
- *Pour tout* $t \in \tau$, $x_3(t) = \begin{cases} -1 \quad +t, & si\ t \in [0,2] \\ 3 \quad -t, & si\ t \in [2,6]. \end{cases}$

τ_3 est finie et limitée, par conséquent l'exécution (τ_3, q_3, x_3) est finie. De plus, (τ_3, q_3, x_3) est maximale.
En effet, supposons qu'il existe une exécution (τ, q, x), telle que $(\tau_3, q_3, x_3) < (\tau, q, x)$. Soit le deuxième intervalle de la trajectoire temporisée τ est de la forme $[2, T]$ (ou $[2, T[$, avec $T > 6$. Soit la trajectoire temporisée τ possède un troisième intervalle. Dans le premier cas, pour tout t in $[2, T[$, $q(t) = 2$, et $x(t) = 3 - t$. Par conséquent, pour $t \in]6, T[$, $x(t) < -3$, et donc $x(t) \notin D_2$.
Dans le deuxième cas, celà signifie qu'il y a un changement de l'état discret à l'instant 6, On a donc $q(6^-) = 2$ et $q(6^+) = 1$. Or $x(6^-) = -3$, et donc $x(6^-) \notin G_{(2,1)}$.
On voit donc que la réciproque de la proposition 2.2 est fausse.
Notons que (τ_2, q_2, x_2) est aussi un préfixe strict de (τ_3, q_3, x_3); c'est d'ailleurs le plus grand commun préfixe de (τ_1, q_1, x_1) et (τ_3, q_3, x_3). Une exécution peut donc être le préfixe de plusieurs exécutions maximales distinctes.

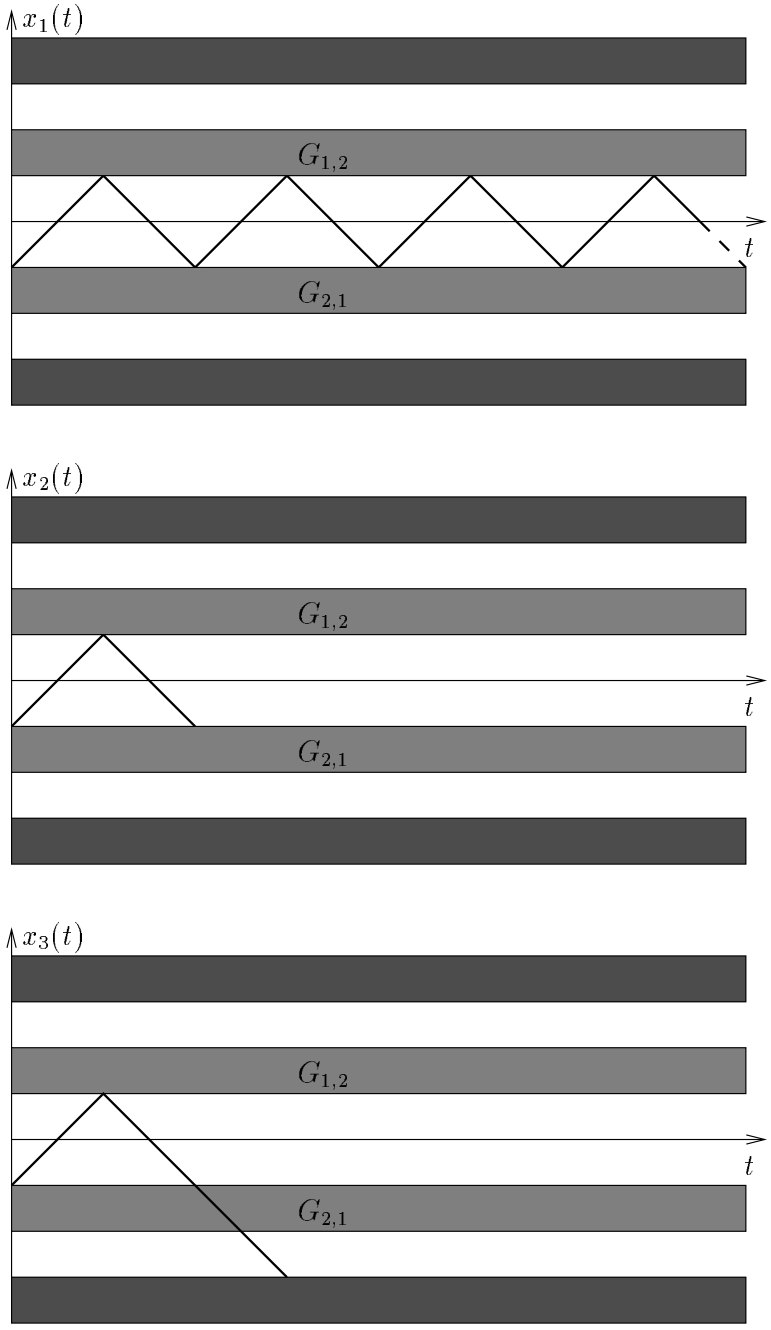


FIG. 10: Exemple d'exécutions hybrides

Cet exemple suscite deux questions :

1. Comment garantir, que toute exécution est le préfixe d'une seule exécution maximale?

2. Sous quelles conditions la réciproque de la proposition 2.2 est elle vrai? Autrement dit, quelles conditions doivent êtres vérifiées pour garantir que toute exécution finie est le préfixe d'une exécution infinie?

Nous examinons ces deux questions dans la partie suivante.

2.2 Systèmes hybrides déterministes et non-bloquants

Les deux questions précédentes nous amènent à définir deux classes de systèmes hybrides, les systèmes hybrides déterministes et les systèmes hybrides non-bloquants.

Définition 2.9 (Système hybride déterministe) *Un système hybride $\mathcal{H}$ est déterministe, si, pour toute condition initiale (t_0, q_0, x_0), $\mathcal{H}$ accepte une unique exécution maximale.*

Définition 2.10 (Système hybride non-bloquant) *Un système hybride $\mathcal{H}$ est non-bloquant, si, pour toute condition initiale (t_0, q_0, x_0), $\mathcal{H}$ accepte une exécution infinie.*

Dans cette partie, on suppose que les équations différentielles définissant la dynamique continue du système sont autonomes. On considère donc un système hybride $\mathcal{H} = \{\mathcal{Q}, \mathcal{E}, \mathcal{D}, \mathcal{F}, \mathcal{G}, \mathcal{R}\}$.
On suppose de plus, que pour tout $q \in \mathcal{Q}$, le champ de vecteur f_q est Lipschitz sur D_q. Ainsi [10], pour tout $x_0 \in D_q$, le problème de Cauchy

$$x'(t) = f_q(x(t)),\ x(t_0) = x_0,\ x(t) \in D_q \tag{2.1}$$

admet une unique solution maximale.
Cette solution est définie sur un intervalle I qui peut être de la forme

- $I = [t_0, +\infty[$;
- $I = [t_0, T]$, avec $T < +\infty$; dans ce cas, $x(T) \in \partial D_q$;
- $I = [t_0, T[$, avec $T < +\infty$; dans ce cas, la limite de $x(t)$ en T existe et

$$\lim_{t \to T} x(t) \in \partial D_q.$$

L'ensemble de ces valeurs limites jouent clairement un rôle essentiel dans le *blocage* du système hybride. Si, lorsque $x(t)$ atteint un de ces points, toute transition est impossible, alors l'exécution du système hybride ne peut se poursuivre.

Définition 2.11 (Points de sortie) *Soit x_0 un point de D_q, tel que $x(t)$, la solution maximale du problème de Cauchy (2.1) est définie sur un intervalle I de longueur finie.*

- *Si $I = [t_0, T]$, alors $x(T)$ est un point de sortie de D_q.*
- *Si $I = [t_0, T[$, alors $\lim_{t \to T} x(t)$ est un point de sortie de D_q.*

L'ensemble des points de sortie est une partie de ∂D_q et est noté S_q.

2.2.1 Caractérisation d'un système hybride déterministe

Lemme 2.1 (Condition nécessaire et suffisante) *Un système hybride est déterministe si et seulement si il vérifie les trois conditions suivantes :*

- *pour tout* $(q,q') \in \mathcal{E}$, $G_{(q,q')} \subseteq S_q$;
- *pour tout* $(q,q') \in \mathcal{E}$ *et* $(q,q'') \in \mathcal{E}$, $q' \neq q''$, $G_{(q,q')} \cap G_{(q,q'')} = \emptyset$;
- *pour tout* $e \in \mathcal{E}$ *et* $x \in G_e$, $R_e(x)$ *contient un unique élément.*

Preuve : Supposons q'une des trois hypothèses n'est pas vérifiée.

−**Premier cas** : Il existe un état (σ, y), une transition $(\sigma,\sigma') \in \mathcal{E}$, tels que $y \in G_{(\sigma,\sigma')}$ et $y \notin S_\sigma$.
Le problème de Cauchy

$$x'(t) = f_\sigma(x(t)),\ x(0) = y,\ x(t) \in D_\sigma$$

admet une solution maximale définie sur un intervalle I de longueur non-nulle (car sinon $y \in S_\sigma$). Par conséquent, $\mathcal{H}$ accepte l'exécution

$$(\{I\}, \{t \mapsto \sigma\}, \{t \mapsto x(t)\}).$$

D'autre part, $\mathcal{H}$ accepte l'exécution

$$(\{[0,0],[0,0]\}, \{t \mapsto \sigma, t \mapsto \sigma'\}, \{t \mapsto y, t \mapsto y'\}) \text{ où } y' \in R_{(\sigma,\sigma')}(y).$$

Aucune de ces deux exécutions n'est un préfixe de l'autre. Pour la condition initiale $(0,\sigma,y)$ $\mathcal{H}$ accepte donc deux exécutions maximales; $\mathcal{H}$ n'est donc pas déterministe.

−**Deuxième cas** : Il existe $(\sigma,\sigma') \in \mathcal{E}$, $(\sigma,\sigma'') \in \mathcal{E}$, tels que $\sigma' \neq \sigma''$ et $G_{(\sigma,\sigma')} \cap G_{(\sigma,\sigma'')} \neq \emptyset$.
Soit y un élément de $G_{(\sigma,\sigma')} \cap G_{(\sigma,\sigma'')}$, $\mathcal{H}$ accepte les exécutions

$$(\{[0,0],[0,0]\}, \{t \mapsto \sigma, t \mapsto \sigma'\}, \{t \mapsto y, t \mapsto y'\}) \text{ où } y' \in R_{(\sigma,\sigma')}(y)$$

et

$$(\{[0,0],[0,0]\}, \{t \mapsto \sigma, t \mapsto \sigma''\}, \{t \mapsto y, t \mapsto y''\}) \text{ où } y'' \in R_{(\sigma,\sigma'')}(y).$$

Pour les mêmes raisons que précédemment, $\mathcal{H}$ n'est pas déterministe.

−**Troisième cas** : Il existe $e = (\sigma,\sigma'), \in \mathcal{E}$, $y \in G_e$, tels que $R_e(y)$ contient plusieurs éléments.
Soient y' et y'' deux éléments distincts de $R_e(y)$, $\mathcal{H}$ accepte les exécutions

$$(\{[0,0],[0,0]\}, \{t \mapsto \sigma, t \mapsto \sigma'\}, \{t \mapsto y, t \mapsto y'\})$$

et

$$(\{[0,0],[0,0]\}, \{t \mapsto \sigma, t \mapsto \sigma'\}, \{t \mapsto y, t \mapsto y''\}).$$

Comme pour les deux premiers cas, on conclut que $\mathcal{H}$ n'est pas déterministe.

Supposons maintenant que $\mathcal{H}$ n'est pas déterministe.

Il existe donc deux exécutions maximales distinctes (τ, q, x) et (τ', q', x') avec la même condition initiale. Soit $(\hat{\tau}, \hat{q}, \hat{x})$ le plus grand commun préfixe de (τ, q, x) et (τ', q', x'). D'après la proposition 2.1, $\hat{\tau}$ est de la forme $\{[\hat{t}_i, \hat{t}_{i+1}]\}_{i=0}^{i=\hat{N}}$. On note $(\sigma, y) = (\hat{q}(\hat{t}_{\hat{N}+1}), \hat{x}(\hat{t}_{\hat{N}+1}))$.

–**Premier cas** : $\hat{t}_{\hat{N}+1} \neq t_{\hat{N}+1}$ ou $\hat{t}_{\hat{N}+1} \neq t'_{\hat{N}+1}$

A l'instant $\hat{t}_{\hat{N}+1}$, il n'y a pas de transitions dans aucune des deux exécutions maximales. Cependant, $x(\hat{t}_{\hat{N}+1}) = x'(\hat{t}_{\hat{N}+1}) = y$, donc par unicité de la solution du problème de Cauchy, $x(t)$ et $x'(t)$ coincident sur un intervalle plus grand que $\hat{I}_{\hat{N}}$. $(\hat{\tau}, \hat{q}, \hat{x})$ n'est donc pas le plus grand commun préfixe de (τ, q, x) et (τ', q', x').

–**Deuxième cas** : $\hat{t}_{\hat{N}+1} = t_{\hat{N}+1}$ et $\hat{t}_{\hat{N}+1} \neq t'_{\hat{N}+1}$ (voir figure 11)

A l'instant $\hat{t}_{\hat{N}+1}$, il y a une transition dans l'exécution (τ, q, x) donc y appartient à une garde du domaine D_σ.

Par contre, il n'y a pas de transition dans l'exécution (τ', q', x'), donc y n'est pas un élément de S_σ.

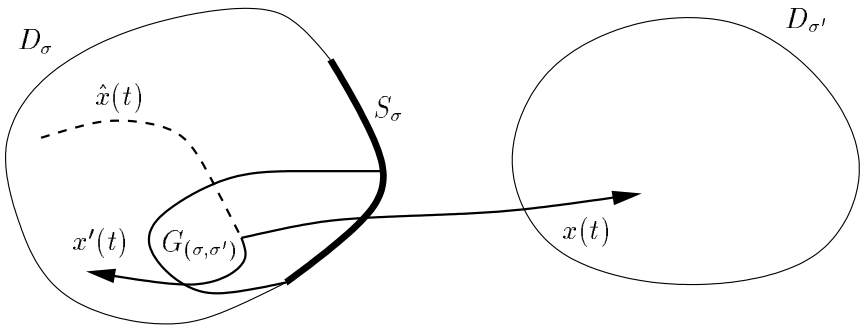


FIG. 11: Système hybride indéterministe (cas 1)

–**Troisième cas** : $\hat{t}_{\hat{N}+1} \neq t_{\hat{N}+1}$ et $\hat{t}_{\hat{N}+1} = t'_{\hat{N}+1}$, symétrique au deuxième cas.

–**Quatrième cas** : $\hat{t}_{\hat{N}+1} = t_{\hat{N}+1}$ et $\hat{t}_{\hat{N}+1} = t'_{\hat{N}+1}$

A l'instant $\hat{t}_{\hat{N}+1}$, il y a une transition dans les deux exécutions. Cependant, comme $(\hat{\tau}, \hat{q}, \hat{x})$ est le plus grand commun préfixe de (τ, q, x) et (τ', q', x'), $(q(t^+_{\hat{N}+1}), x(t^+_{\hat{N}+1})) \neq (q'(t^+_{\hat{N}+1}), x'(t^+_{\hat{N}+1}))$.

Si $q(t^+_{\hat{N}+1}) \neq q'(t^+_{\hat{N}+1})$, alors, nécessairement $y \in G_{(\sigma, q(t^+_{\hat{N}+1}))} \cap G_{(\sigma, q'(t^+_{\hat{N}+1}))}$ (voir figure 12).

Sinon, si $x(t^+_{\hat{N}+1}) \neq x'(t^+_{\hat{N}+1})$, alors, $R_{(\sigma, q(t^+_{\hat{N}+1}))}(y)$ contient au moins deux éléments (voir figure 13).

■

Pratiquement, la condition du lemme 2.1 signifie qu'une transition n'est possible que si l'évolution continue est bloquée. De plus, en chacun de ces points une seule transition est possible et la variable continue ne peut être réinitialisée qu'à une seule valeur.

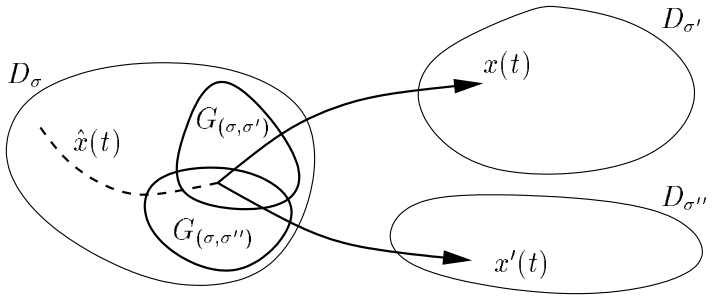


FIG. 12: Système hybride indéterministe (cas 2)

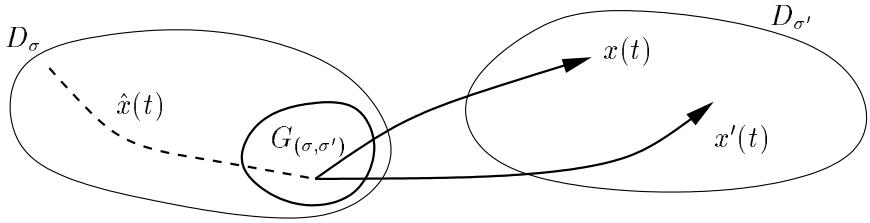


FIG. 13: Système hybride indéterministe (cas 3)

2.2.2 Caractérisation d'un système hybride non-bloquant

Lemme 2.2 (Condition suffisante) *Si*

$$\forall q \in \mathcal{Q},\ \forall x \in S_q,\ \exists (q,q') \in \mathcal{E}, x \in G_{(q,q')},$$

alors le système hybride est non-bloquant.

Preuve : Supposons qu'il existe une condition initiale pour laquelle $\mathcal{H}$ n'accepte pas d'exécution infinie. Soit (τ, q, x) une exécution maximale partant de cette condition initiale. L'exécution est finie et donc $\tau = \{I_i\}_{i=0}^{i=N}$ est une trajectoire temporisée finie et limitée. On a également $q = \{q_i\}_{i=0}^{i=N}$ et $x = \{x_i\}_{i=0}^{i=N}$. Notons $\sigma = q_N(t_N)$ et $y = x_N(t_N)$.

On considère le problème de Cauchy,

$$\hat{x}'_N(t) = f_\sigma(\hat{x}_N(t)),\ \hat{x}_N(t_N) = y,\ \hat{x}_N(t) \in D_\sigma.$$

La solution maximale $\hat{x}_N$ est définie sur un intervalle $\hat{I}_N$. Si $\hat{I}_N = [t_N, T[$, avec $T < +\infty$ alors

$$\lim_{t \to T} \hat{x}_N(t) \in S_\sigma.$$

Donc la limite de $\hat{x}_N$ en T appartient à une garde du domaine D_σ et donc à D_σ. Par conséquent, $\hat{x}_N$ peut être prolongée en T et n'est donc pas solution maximale du problème de Cauchy. Nécessairement, $\hat{I}_N$ est de la forme $[t_N, T]$ ou $[t_N, +\infty[$.

On a $I_N \subseteq \hat{I}_N$ et pour tout $t \in I_N$, $\hat{x}_N(t) = x_N(t)$.

– Si $I_N \neq \hat{I}_N$ (voir figure 14), on définit

$$\begin{cases} \hat{\tau} &= \{\{I_i\}_{i=0}^{i=N-1}, \hat{I}_N\}; \\ \hat{q} &= \{\{q_i\}_{i=0}^{i=N-1}, \hat{q}_N\}, \quad \text{avec } \hat{q}_N(t) = \sigma \text{ sur } \hat{I}_N; \\ \hat{x} &= \{\{x_i\}_{i=0}^{i=N-1}, \hat{x}_N\}. \end{cases}$$

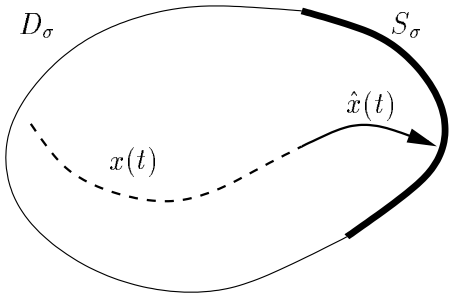


FIG. 14: Prolongement de la trajectoire d'un système hybride non-bloquant (cas 1)

– Si $\hat{I}_N = I_N$ (voir figure 15), alors nécessairement I_N est de la forme $[t_N, t_{N+1}]$ car τ est limitée. De plus, dans ce cas $x_N(t_{N+1}) = \hat{x}_N(t_{N+1})$ est un élément de $S_{q_N(t_N)}$. Il existe donc une transition $e = (\sigma, \sigma')$ telle que $x_N(t_{N+1}) \in G_e$. On définit donc

$$\begin{cases} \hat{\tau} &= \{\{I_i\}_{i=0}^{i=N}, [t_{N+1}, t_{N+1}]\}; \\ \hat{q} &= \{\{q_i\}_{i=0}^{i=N}, q_{N+1}\}, & \text{avec } q_{N+1}(t_{N+1}) = \sigma'; \\ \hat{x} &= \{\{x_i\}_{i=0}^{i=N}, x_{N+1}\}. & \text{avec } x_{N+1}(t_{N+1}) \in R_e(x_N(t_{N+1})). \end{cases}$$

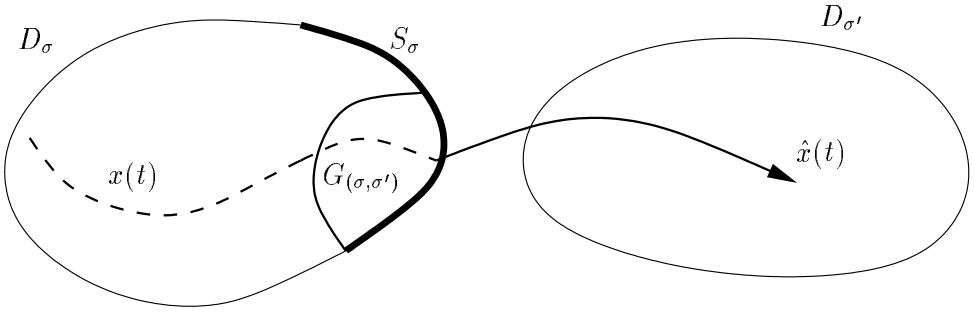


FIG. 15: Prolongement de la trajectoire d'un système hybride non-bloquant (cas 2)

Dans les deux cas, (τ, q, x) est un préfixe strict de $(\hat{\tau}, \hat{q}, \hat{x})$ et le système hybride accepte l'exécution $(\hat{\tau}, \hat{q}, \hat{x})$ donc (τ, q, x) ne peut pas être maximale.

■

De manière intuitive, la condition donnée dans le lemme 2.2 signifie qu'à chaque fois que l'évolution continue du système est bloquée (quand la variable continue atteint la frontière du domaine et est obligée de sortir de celui-ci) alors un changement d'état discret est possible.
Cette condition n'est pas nécessaire. En effet, considérons le système hybride $\mathcal{H}$ donné par

1. $\mathcal{Q} = \{1, 2\}$
2. $\mathcal{E} = \{(1, 2)\}$
3. $D_1 = [0, 1[, D_2 = \mathbb{R}$
4. $f_1(x) = 1,\ f_2(x) = -1$
5. $G_{(1,2)} = [0, 1[$
6. $R_{(1,2)} = \{x\}$

On a $S_1 = 1$, donc les hypothèses du lemme 2.2 ne sont pas vérifiées. Soit (t_0, σ_0, y_0) une condition initiale.
Si $\sigma_0 = 2$ alors l'exécution

$$(\{[t_0, +\infty[\}, \{t \mapsto 2\}, \{t \mapsto y_0 + t - t_0\})$$

est infinie et acceptée par $\mathcal{H}$.
Si $\sigma_0 = 1$ alors l'exécution

$$(\{[t_0, t_0], [t_0, +\infty[\}, \{t \mapsto 1, t \mapsto 2\}, \{t \mapsto y_0, t \mapsto y_0 + t - t_0\})$$

est infinie et acceptée par $\mathcal{H}$.
Donc, pour toute condition initiale, le système hybride accepte une exécution infinie; $\mathcal{H}$ est non-bloquant.

Cependant, pour un système hybride déterministe, la condition du lemme 2.2 devient nécessaire.

Lemme 2.3 (Condition nécessaire et suffisante) *Un système hybride déterministe est non-bloquant si et seulement si*

$$\forall q \in \mathcal{Q},\ \forall x \in S_q,\ \exists (q, q') \in \mathcal{E}, x \in G_{(q,q')}.$$

Preuve : Supposons qu'il existe un état (σ, y) tel que $y \in S_\sigma$ et pour tout $(\sigma, \sigma') \in \mathcal{E}$, $y \notin G_{(\sigma,\sigma')}$.
Il existe $y_0 \in D_\sigma$ tel que la solution maximale du problème de Cauchy

$$x'(t) = f_\sigma(x(t)),\ x(0) = y_0,\ , x(t) \in D_\sigma$$

soit définie sur un intervalle I de la forme $[0, T]$ ou $[0, T[$, avec $T < +\infty$ et tel que

$$\lim_{t \to T} x(t) = y.$$

Ainsi, le système hybride accepte l'exécution $(\{I\}, \{t \mapsto \sigma\}, \{t \mapsto x(t)\})$. Clairement, cette exécution est finie et maximale.
De plus, pour la condition initiale $(0, \sigma, y_0)$, c'est l'unique exécution maximale car le système hybride est déterministe. Donc, le système hybride n'accepte pas d'exécution infinie pour la condition initiale $(0, \sigma, y_0)$ et n'est donc pas non-bloquant.

■

2.2.3 Existence et unicité des exécutions

Théorème 2.1 (Existence et unicité des exécutions) *Soit un système hybride $\mathcal{H}$ vérifiant les trois conditions suivantes*

- *pour tout $q \in \mathcal{Q}$ $\bigcup_{(q,q')} G_{(q,q')} = S_q$;*
- *pour tout $(q, q') \in \mathcal{E}$ et $(q, q'') \in \mathcal{E}$, $G_{(q,q')} \cap G_{(q,q'')} = \emptyset$;*
- *pour tout $e \in \mathcal{E}$ et $x \in G_e$, $R_e(x)$ contient un unique élément.*

Alors, pour toute condition initiale (t_0, σ_0, y_0), $\mathcal{H}$ accepte une unique exécution infinie (τ, q, x). Toute exécution de $\mathcal{H}$ ayant (t_0, σ_0, y_0) pour condition initiale est un préfixe de (τ, q, x).

Preuve : D'après le lemme 2.3, $\mathcal{H}$ est non-bloquant. Par conséquent, pour toute condition initiale (t_0, σ_0, y_0) il existe une exécution infinie (τ, q, x).
D'après le lemme 2.1, $\mathcal{H}$ est déterministe. Donc, pour toute condition initiale il existe une unique exécution maximale.
D'après la proposition 2.2, toute exécution infinie est maximale donc pour toute condition initiale il existe une unique exécution infinie.
De plus, comme $\mathcal{H}$ est déterministe, toute éxecution ayant (t_0, σ_0, y_0) pour condition initiale est un préfixe de (τ, q, x).

■

Ainsi, l'existence et l'unicité d'une exécution infinie pour toute condition initiale est garantie par des critères simples. Le calcul des ensembles $\{S_q, q \in \mathcal{Q}\}$ constitue la principale difficulté de vérification de ces critères. Le lecteur pourra consulter l'article [18].

2.3 Exécution Zénon

On suppose que les hypothèse du théorème 2.1 sont vérifiées. Ainsi, pour toute condition initiale, le système hybride $\mathcal{H}$ accepte une unique exécution infinie (τ, q, x). La trajectoire temporisée τ peut donc être :

- finie illimitée, le système hybride effectue un nombre finie de transitions en un temps infini;
- infinie illimitée, le système hybride effectue une infinité de transitions en un temps infini;
- infinie limitée, le système hybride effectue une infinité de transitions en un temps fini.

Dans le dernier cas, on dit que l'exécution (τ, q, x) est Zénon.

Définition 2.12 (Exécution Zénon) *Une exécution (τ, q, x) est Zénon si τ est infinie et limitée.*

Références

[1] A.A. Andronov, A.A. Vitt, S.E. Khaikin, Theory of oscillators, Pergamon, 1966.

[2] E. Asarin, O. Bournez, T. Dang, O. Maler, Approximate reachability analysis of piecewise linear dynamical systems, *Hybrid Systems : Computation and Control*, N. Lynch et B. H. Krogh (Eds), no. 1790 in LNCS, pp 21-31, Springer, 2000.

[3] E. Asarin, T. Dang, A. Girard, Reachability of non-linear systems using conservative approximations, *Hybrid Systems : Computation and Control*, O. Maler et Amir Pnueli (Eds), no. 2623 in LNCS, pp 22-35, Spinger, 2003.

[4] T. Bak, J. Bendtsen, A.P. Ravn, Hybrid control design for a wheeled mobile robot, *Hybrid Systems : Computation and Control*, O. Maler et Amir Pnueli (Eds), no. 2623 in LNCS, pp 50-65, Spinger, 2003.

[5] C. Belta, J. Schug, T. Dang, V. Kumar, M. Mintz, G.J. Pappas, H. Rubin, P. Dunlop, Stability and reachability analysis of a hybrid model of luminescence in the marine bacterium Vibrio fischeri, *Proc. of the 40th IEEE CDC*, pp 869-874, 2001.

[6] A. Bemporad, P. Borodani, M. Manneli, Hybrid control of an automotive robotized gearbox for reduction of consumptions and emissions, *Hybrid Systems : Computation and Control*, O. Maler et Amir Pnueli (Eds), no. 2623 in LNCS, pp 81-96, Spinger, 2003.

[7] M.S. Branicky, V.S. Borkar, S.K. Mitter, A unified framework of hybrid control : model and optimal control theory, *IEEE Trans. Automatic Control*, 43(1):31-45, 1998.

[8] L.O. Chua, A.C. Dang, Canonical piecewise-linear modeling, *IEEE Trans. Circuits and Systems*, 33(5):511-525, 1986.

[9] J. Della Dora, A. Maignan, M. Mirica-Ruse, S. Yovine, Hybrid computation, *Proc. ISSAC'01*, pp 101-108, 2001.

[10] J. Dieudonné, Calcul infinitésimal, *Collection Méthodes*, Hermann, 1980.

[11] J.G. Dumas, A. Rondepierre, Modeling the electrical activity of a neuron by a continuous and piecewise linear hybrid system, *Hybrid Systems : Computation and Control*, O. Maler et Amir Pnueli (Eds), no. 2623 in LNCS, pp 22-35, Spinger, 2003.

[12] N.H. El-Farra and P.D. Christofides, Switching and feedback laws for control of constrained switched nonlinear systems, *Hybrid Systems : Computation and Control*, C.J. Tomlin, M.R. Greenstreet (Eds), no. 2289 in LNCS, pp 164-178, Springer, 2002.

[13] A.F. Filippov, Differential equations with discontinuous right-hand side, Kluver Academic Publishers, 1988.

[14] M. Garavello, B. Piccoli, Hybrid necessary principles: an application to a car with gears, *Proc. IFAC conference ADHS 2003*, Elsevier, 2003. (à paraitre)

[15] A. Girard, Approximate solutions of ODEs using piecewise linear vector fields, *Proc. CASC 2002*, pp 107-120, 2002.

[16] A. Girard, Computation and stability analysis of limit cycles in piecewise linear hybrid systems, *Proc. IFAC conference ADHS 2003*, Elsevier, 2003. (à paraitre)

[17] K.H. Johansson, J. Lygeros, S. Sastry, M. Egerstedt, Simulation of Zeno hybrid automata, *Proc. IEEE conference CDC*, 1999.

[18] J. Lygeros, K.H. Johansson, S. Sastry, M. Egerstedt, On the existence of executions of hybrid automata, *Proc. IEEE conference CDC*, 1999.

[19] A.S. Matveev, A.V. Savkin, Qualitative theory of hybrid dynamical systems, Birkhäuser, 2000.

[20] M. Mirica-Ruse, Contribution à l'étude des systèmes hybrides, *Thèse de Doctorat*, Université Joseph Fourier, Grenoble, 2002.

[21] A.M. Samoilenko, N.A. Perestyuk, Impulsive differential equations, *Series on Nonlinear Science*, World Scientific, 1995.

[22] S. Simic, K. H. Johansson, S. Sastry, J. Lygeros, Towards a geometric theory of hybrid systems, *Hybrid Systems : Computation and Control*, N. Lynch et B. H. Krogh (Eds), no. 1790 in LNCS, pp 421-436, Springer, 2000.

[23] C. Tomlin, G.J. Pappas, J. Lygeros, D.N. Godbole, S. Sastry Hybrid control models of next generation air traffic management, *Hybrid Systems IV*, no 1273 in LNCS, Springer 1997.

Exposés Jeunes Chercheurs

Hypergraphes : Combinatoire Enumérative et hypergraphes aléatoires

Tsiry Andriamampianina

Nous énumérons les hypergraphes b-uniformes connexes, suivant les paramètres n et m désignant respectivement le nombre de sommets et le nombre d'hyperarêtes. Nous utiliserons pour faire cette énumération, les séries génératrices : les séries génératrices exactes des hypergraphes b-uniformes connexes d'excès k.
Nous établirons alors des résultats d'énumérations asymptotiques pour les hypergraphes à n sommets.

Timed Mirroring Typed Decision Graphs : Un Modèle de Structure Symbolique pour la Vérification des Systèmes Temps-Réel

Syrine Ayadi

Nous proposons une structure symbolique pour la vérification (model checking) des systèmes temporisés temps-réel : les TMTDGs (Timed Mirroring Typed Decision Graphs), qui représente, d'une part, structurellement, une version réduite des BDDs, en exploitant la propriété du « miroir » et d'autre part, sémantiquement, les contraintes temporelles du système. Cette approche permet d'aboutir à des gains considérables en espace mémoire.

Une méthode de factorisation absolue des polynomes en deux variables

Guillaume Cheze

Donner la factorisation absolue d'un polynôme appartenant à Q[X,Y] signifie donner tous ses facteurs irréductibles dans C[X,Y]. Nous verrons dans cet exposé un moyen d'obtenir une telle factorisation.

Réseaux de régulation génétique affines par morceaux

Etienne Farcot

On présentera un modèle des phénomènes d'interaction et de régulation génétique. Ce modèle se formule comme un système d'équations différentielles affines par morceaux, ce qui permet certains traitements formels pour la simulation et l'analyse, qui seront précisés durant l'exposé.

LinBox : une bibliothèque générique pour l'algèbre linéaire exacte

Pascal Giorgi

Le projet LinBox (action internationale USA, Canada, France) a pour but de développer une bibliothèque C++ générique et efficace pour les problèmes fondamentaux de l'algèbre linéaire exacte. La généricité de la bibliothèque LinBox est basée sur la définition d'interfaces que les différentes composantes logicielles doivent respecter. Ces interfaces ont permis de définir des "wrappers" efficaces basés sur des bibliothèques externes spécialisées pour l'arithmétique de corps finis (NTL, Givaro,...) et pour les routines de base de l'algèbre linéaire (BLAS).
Nous présenterons dans un premier temps nos travaux sur la mise en place d'arithmétique des corps finis dans la bibliothèque LinBox puis nous aborderons les différents concepts algorithmiques developpés dans la bibliothèque.

Pseudozéros de polynômes : théorie et applications

Stef Graillat

Les pseudo-zéros d'un polynôme donné P sont les zéros des polynômes proches de P au sens d'une certaine norme.

Nous donnons des applications de cette notion en calcul symbolique-numérique et en robustesse en théorie du contrôle.

Exploration massivement distribuée du graphe d'Internet : une approche expérimentale

Jean-Loup Guillaume

Les cartes du réseau Internet sont généralement construites en utilisant l'outil traceroute depuis quelques sources vers beaucoup de destinations. Il est apparu récemment que cette méthode d'exploration donne une vision non seulement partielle mais aussi biaisée de la topologie réelle d'Internet. Ces constatations ont fait émerger l'idée qu'il faudrait augmenter le nombre de sources pour améliorer la qualité des cartes. L'objectif de cet exposé est de présenter un ensemble d'expériences destinées à évaluer la pertinence d'une telle approche.
Il apparait que les propriétés statistiques du réseau ont une grande influence sur la qualité des cartes obtenues, lesquelles peuvent être améliorées en utilisant une exploration massivement distribuée.

Certification d'un algorithme d'élimination des quantificateurs sur $\mathbb{R}$

Assia Mahboubi

L'algorithme d'Hormander est un algorithme très élémentaire d'élimination des quantificateurs sur les corps réels clos, au sens où il ne fait intervenir que des propriétés très simples de ces corps comme le TVI pour les polynômes et des objets facilement représentables en machine comme des tableaux de signes. C'est pour cela qu'il a été choisi comme support pour programmer dans le système Coq (assistant à la preuve basé sur le calcul des constructions inductives) une "tactique" de décision des systèmes polynomiaux sur $\mathbb{R}$. On présentera ici l'algorithme d'Hormader et les différents aspects théoriques et pratiques liés à l'implantation en Coq d'une telle procédure.

Approximation d'un système dynamique jusqu'à l'ordre 2

Cyrille Martig

Nous présentons un algorithme d'identification de systèmes dynamiques à une entrée avec dérive, basé sur l'acquisition numérique des coefficients de la série génératrice associée jusqu'à l'ordre 2, à partir de jeux d'entrée-sortie éventuellement bruités. Le calcul des coefficients nécessite la connaissance des dérivées successives des fonctions d'entrée-sortie, obtenues grâce aux polynômes de Lagrange.

La recherche d'information conceptuelle

Ibtissem Nafkha

Nous avons proposé un système de recherche d'information conceptuel basé sur l'analyse de concept formel. le système proposé utilise une approche conceptuelle dans le processus de recherche pour trouver les concepts à partir de différentes bases documentaires. Le système de recherche d'information coopératif est constitué d'un ensemble de systèmes de recherche d'information qui coopèrent entre eux pour traiter une requête. Chaque système de recherche d'information possède une base de documents locale sur laquelle nous appliquons la connexion de Galois pour retrouver les documents répondant à la requête. Pour traiter une requête, chaque système de recherche d'information exécute l'algorithme de recherche proposé sur sa base documentaire locale. Ainsi, le résultat de chaque système de recherche d'information est un concept dont le domaine est les termes cherchés et son codomaine est les documents trouvés. En se basant sur l'ensemble de concepts trouvés à partir de différentes bases documentaires, nous formulons la réponse finale en appliquant deux algorithmes proposés.

Calcul du polynôme caractéristique sur des corps finis

Clément Pernet

Nous traiterons dans cet exposé d'un des problèmes classiques en algèbre linéaire : le calcul du polynôme caractéristique et plus généralement, celui de la forme normale de Frobenius d'une matrice. Nous nous intéresserons plus particulièrement aux matrices denses à coefficients dans des corps finis.
Dans une première partie, nous introduirons nos résultats concernant la triangularisation de matrices denses. Une nouvelle approche utilisant les puissantes routines numériques des BLAS pour le calcul exact a permis d'atteindre des performances jusque là inégalées. Ces routines efficaces en pratique serviront de brique de base pour les algorithmes de calcul du polynôme caractéristique.
Dans une deuxième partie, nous présenterons deux algorithmes pour le calcul du polynôme caractéristique, basés sur des triangularisation par blocs. Le premier est inspiré des méthodes de Danilevski et de Krylov. Nous en présentons une version par blocs, le rendant très efficace en pratique. Le second est celui de Keller-Gehrig, pour lequel nous avons montré que l'usage de la factorisation LSP est particulièrement adaptée. Enfin une comparaison des performances pratiques de ces deux algorithmes montrera que leurs domaines de prédilection sont complémentaires et dépendent du nombre de sous espaces invariants de la forme de Frobenius associée.

Méthodes hybrides : étude de l'activité électrique d'un neurone

Aude Rondepierre

Dans cet exposé, nous proposons un système hybride pour modéliser le potentiel électrique émis par un neurone en réponse à une stimulation électrique extérieure. Expérimentalement Hodgkin et Huxley ont construit un système dynamique non linéaire de dimension 4 pour simuler cette activité. Cependant l'analyse mathématique est complexe et les résultats sont essentiellement numériques.
L'idée développée dans cet exposé est d'utiliser une approximation affine par morceaux, continue comme modèle hybride de la dynamique de Hodgkin-Huxley. Nous verrons que notre modèle reproduit avec précision les caractéristiques du modèle initial et qu'il permet de surcroit un calcul explicite des solutions.

Génération de code parallèle à partir d'une fonction de placement

Yosr Slama

La dernière étape de la parallélisation automatique des programmes est la génération de code. Beaucoup de travaux sont fait pour générer du code parallèle à partir d'une fonction d'ordonnancement (temps), cependant, très peu de travaux existent pour la génération de code à partir d'un placement (les processeurs). Ce dernier présentant beaucoup d'avantages par rapport à l'ordonnacement, nous proposons une approche et un outil de génération de programmes parallèles valides respectant un placement de calcul donné. Ceci consiste essentiellement en 2 phases : la génération des boucles parallèles et la génération des synchronisations.

Algorithme récursif binaire pour le calcul de pgcd

Damien Stehle

Les algorithmes usuels de PGCD, comme l'algorithme d'Euclide et l'algorithme binaire, ont des complexités quadratiques en la taille des entiers considérés. En 1970, Knuth a proposé un algorithme en temps quasi-linéaire (de complexité $O(nlog^2loglog(n))$, cette borne a été prouvée par Schonhage), qui repose sur la multiplication rapide d'entiers, à base de FFT. Cet algorithme est une variante récursive de l'algorithme d'Euclide. Malgré sa bonne complexité asympotique, il n'est intéressant que pour de très grands nombres, à cause de l'utilisation interne d'une routine très coûteuse de "réparation locale" des quotients calculés. Dans cet exposé, nous présenterons un algorithme récursif basé non pas sur la division Euclidienne classique mais sur une nouvelle division binaire. Il a la même structure et la même complexité asymptotique que l'algorithme de Knuth, mais il n'y a plus besoin de procédure de "réparation locale". Cela a deux avantages : la preuve de correction de l'algorithme est plus simple à établir, et d'un point de vue pratique, l'implantation est plus facile et plus efficace.

Modèles hybrides de systèmes de régulation biologiques

Laurent Tournier

Les systèmes de régulation issus de la biologie cellulaire sont des systèmes complexes, présentant à la fois une grande diversité de comportements et une certaine robustesse. Il existe dans la littérature plusieurs approches possibles pour comprendre et modéliser de tels phénomènes (systèmes dynamiques discrets, systèmes différentiels, hybrides etc.). Nous verrons à travers l'étude de l'opéron lactose (Jacob & Monod, 1961) en quoi consiste un réseau de régulation génétique et comment en proposer un modèle.

Index des participants

Stehle Damien
Stouls Nicolas

T

Tecourt Jean-Pierre
Tisserand Arnaud
Tournier Laurent
Trystram Denis

V

Varouchas Georges
Veyrat Charvillon Nicolas
Vienne Jerome

Z

Zimmermann Paul

Index des exposants