Quelques propositions pour la mise en oeuvre d'algorithmes combinatoires

Didier Tallot

Académie de Montpellier

UNIVERSITE DES SCIENCES ET TECHNIQUES DU LANGUEDOC

THESE

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le grade de

DOCTEUR DE 3ème CYCLE
Option INFORMATIQUE

QUELQUES PROPOSITIONS POUR LA MISE EN OEUVRE D'ALGORITHMES COMBINATOIRES

par

Didier TALLOT

Soutenue à Montpellier le 28 juin 1985
devant la commission d'examen :

Président	M. M. CHEIN
Examinateurs	MM. C. BOKSENBAUM
	A. CAZES
	O. COGIS
	J. FERRIE
	M. HABIB
	J.M. LABORDE

Académie de Montpellier

UNIVERSITE DES SCIENCES ET TECHNIQUES DU LANGUEDOC

THESE

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le grade de

DOCTEUR DE 3ème CYCLE
Option INFORMATIQUE

QUELQUES PROPOSITIONS POUR LA MISE EN OEUVRE D'ALGORITHMES COMBINATOIRES

par

Didier TALLOT

Soutenue à Montpellier le 28 juin 1985
devant la commission d'examen :

Président	M. M. CHEIN
Examinateurs	MM. C. BOKSENBAUM
	A. CAZES
	O. COGIS
	J. FERRIE
	M. HABIB
	J.M. LABORDE

REMERCIEMENTS

Je voudrais tout d'abord remercier Monsieur le professeur Michel HABIB, pour m'avoir accueilli au Departement Informatique Appliquée de l'Ecole Nationale Supérieure des Mines de Saint-Etienne pour diriger mes recherches. Son intérêt et son aide ont toujours été extrêmement précieux.

Je tiens à remercier Monsieur le professeur CHEIN qui m'a fait l'honneur d'accepter de présider le jury de cette thèse.

Je remercie vivement Messieurs Jean FERRIE et Claude BOKSENBAUM, professeurs, et Messieurs Alain CAZES et Olivier COGIS, maitre-assistants, d'avoir accepté de juger ce travail.

Monsieur Jean-marie LABORDE, chargé de recherche à l'IMAG de Grenoble, responsable avec Michel HABIB du projet CABRI, a bien voulu se déplacer pour participer à ce jury. Qu'il en soit remercié.

Cette thèse, considéré comme le travail d'un seul, n'aurai pu être menée à bien sans l'aide des personnes suivantes.

Messieurs Michel DAO et Jean-Philippe RICHARD, chercheurs au CNET/PARIS, pour leurs aides et leurs amitiés.

Mes collégues de l'équipe "Algorithmique et Programmation", Ehoud ARONOVITZ, Vincent BOUCHITTE, Mohamed HAMROUN, Roland JEGOU pour toutes les discussions jusqu'à tres tard le soir.

Je voudrais aussi exprimer mon amitié à tous les gens du département, avec qui de fréquents échanges de points de vue ont été profitables.

Je remercie enfin Messieurs DARLES, LOUBET et VELAY pour leur participation avec beaucoup de soin et de patience, à la réalisation de cet ouvrage.

Ce rapport a été réalisé dans un environnement UNIX grâce à l'utilitaire de traitement de texte GROFF (Girardot's Ridiculous Output File Formatter) dévelloppé par J.J. GIRARDOT, que je tiens ici à remercier.

Toutes erreurs de typographie ne peuvent donc être dues qu'au logiciel ou à la quincaillerie.

INTRODUCTION GENERALE

PREMIERE PARTIE : SUR L'ALGORITHMIQUE GEOMETRIQUE

INTRODUCTION

1. QUELQUES MOTS SUR LES TYPES ABSTRAITS

2. REFERENCES

CHAPITRE 1 : ALGORITHME DE CALCUL D'ENVELOPPES CONVEXES

1. INTRODUCTION

2. ENVELOPPES CONVEXES DANS LE PLAN

2.1. LA FONCTION CONVEX

2.2. OPERATION "POINTS_EXTREM"

2.3. ANALYSE EN COMPLEXITE

2.4. ALGORITHME D'ENVELOPPE CONVEXE

2.5. CAS DE DEGENERESCENCE

3. REFERENCES

CHAPITRE 2 : BALAYAGE DU PLAN

1. INTRODUCTION

2. ORDRE SUR LE PLAN ET PRINCIPE DU BALAYAGE

2.1. DEFINITION DE L'ORDRE SUR LES SEGMENTS

2.2. STRUCTURE DE DONNEES ET ALGORITHME

3. INTERSECTIONS DE SEGMENTS DANS LE PLAN

3.1. BORNE INFERIEURE

3.2. CALCUL DES INTERSECTIONS

3.3. UN AUTRE PRINCIPE SANS CALCUL D'INTERSECTION

3.4. CAS PARTICULIER

3.4.1. ensemble de segments avec deux directions

3.4.2. généralisation à un nombre constant de directions

4. EXTENSIONS DU BALAYAGE DU PLAN A D'AUTRE OBJETS GEOMETRIQUES

5. CONCLUSIONS

6. REFERENCES

ANNEXE 1 : GENERALISATION DE L'ALGORITHME DU DRAPEAU HOLLANDAIS

ANNEXE 2 : LE PARADIGME DU RATISSAGE

DEUXIEME PARTIE : SUR LA MANIPULATION INTERACTIVE DE GRAPHES

INTRODUCTION : UN OUTIL DE MANIPULATION DE GRAPHES

1. HISTORIQUE ET MOTIVATIONS

2. PLAN

CHAPITRE 1 : DEVELOPPEMENT DE LOGICIELS INTERACTIFS

1. ETUDE DU CABRI DE MONTPELLIER

2. INTERFACE HOMME-MACHINE

3. MODULARITE

4. EXTENSIBILITE

4.1. INTRODUCTION

4.2. EXTENSIBILITE DANS L'EDITION DE DESSIN

4.3. EXTENSIBILITE DANS LES FONCTIONS DE CALCUL

CHAPITRE 2 : DESCRIPTION DE KABRI

1. ANALYSE DES TACHES ET DES BESOINS

1.1. DEFINITION DES PRINCIPAUX TERMES

1.2. TACHE PRINCIPALE

1.3. MANIPULATIONS DE DESSINS DE GRAPHES

1.4. CALCUL DE PROPRIETES ET D'INVARIANTS

1.5. ECRITURE DE NOTES DE TRAVAIL

1.6. ARCHIVAGE

1.7. FABRICATION DE TEXTES DE RAPPORT

2. QUELQUES REMARQUES

3. DESCRIPTION SEMANTIQUE

3.1. MODELE POUR L'UTILISATEUR

3.1.1. description des objets manipulés

3.1.2. description des opérations

4. DESCRIPTION SYNTAXIQUE

4.1. REPRESENTATION DES DONNEES

4.1.1. objets intrinsèques

4.1.2. objets de contrôle

4.2. DESCRIPTION DES PERIPHERIQUES

4.3. DESCRIPTION DES COMMANDES

5. ENVIRONNEMENT LOGICIEL D'IMPLANTATION

6. POURQUOI LISP?

CHAPITRE 3 : CONCLUSIONS

ANNEXE 1 : COMMUNICATIONS ENTRE PROCESSUS AVEC UNIX

ANNEXE 2 : BIBLIOGRAPHIE SUR L'INTERACTIVITE

Introduction générale

Le travail, exposé dans ce rapport, se divise en deux parties.

La première partie a fait l'objet d'un rapport de recherche publié en Avril 1984 [Tall]. La deuxième partie résulte d'un travail qui sést déroulé de juin 1984 à juin 1985.

La première partie contient un exposé sur deux problèmes en complexité géométrique.

Pour produire des images de hautes qualités, on est obligé de manipuler des grandes quantités de données. D'où l'intérêt d'étudier des algorithmes et des structures de données efficaces pour résoudre les problèmes géométriques. On pourra ainsi obtenir des méthodes efficaces pour la manipulation de données graphiques.

On peut citer à ce propos, les travaux pionners de SHAMOS dans le domaine de la complexité géomètrique [Sham].

Le deux problèmes abordés ici sont:

- le calcul de l'enveloppe convexe d'un ensemble de points.
- le calcul des intersections d'un ensemble de segments.

Les méthodes algorithmiques présentées illustrent deux principes généraux de conception d'algorithmes:

- le principe "Divide-and-Conquer".
- le principe du ratissage.

L'utilisation des types abstraits pour l'élaboration et la description des algorithmes présenté nous a permis:

- un exposé que nous espérons plus clair
- des preuves simple indépendantes de la représentation des données
- des analyses en complexité pour différentes "implémentation" des structures de données
- de découvrir certaines faiblesses des algorithmes exposés dans les premiers articles sur le balayage du plan
- de trouver un algorithmes généralisant une partie de ceux connus pour le calcul de l'enveloppe convexe

Les travaux de BARBERYE et JOUBERT [Barb] sur l'outil interactif OASIS nous ont indiqués les types abstraits comme une méthode de spécification d'algorithmes.

Le premier chapitre expose un algorithme de calcul d'enveloppes convexes géomètriques d'un ensemble de n points dans le plan.

Cet algorithme est une application du principe récursif "Divide-and-conquer". Sa complexité et O(|C|n), où |C| est le nombre de points de l'enveloppe convexe C calculé.

Cet algorithme propose un formalisme recouvrant et généralisant des algorithmes déjà connus.

Le deuxième algorithme étudié est celui du **balayage du plan** qui est une application du paradigme de **ratissage.**

L'application du balayage du plan a de nombreuses applications pratiques, en CAD (par exemple en VLSI), en imagerie par ordinateur...

Il permet d'obtenir des solutions à de nombreux problèmes, du types intersections, identifications, renseignements sur la topologie de figures formés d'un ensemble de segments de droite.

La complexité de cet algorithme est en O((n+s)logn), où n est le nombre de segments et s le nombre, inconnu initialement, des intersections. n mesure, en fait, la taille de la donnée et s sa complexité (et la taille des résultats).

La complexité des algorithmes d'intersections de segments est en général en $O(n^2)$ (test sur chaque paire de segments). Le plus mauvais cas, pour le balayage du plan, est $O(n^2 \log n)$ ce qui le rend à première vue peu intéressant. Mais les configurations où $s=O(n^2)$ sont rares, en pratique s est égal à O(n) pour des configurations réalistes. Ce qui donne une complexité en O(nlogn) semblable à celle du tri.

Un exposé, que nous espérons clair, de l'algorithme et de ses propriétés est donné dans le chapitre 2. Nous y étudions différentes structures de données possibles et les complexités de l'algorithme pour chacunes de ces structures.

Ces méthodes ont été employées avec succés dans des logiciels par MM. MICHELUCCI et GANGNET [Gang] et par MOISSINAC [Mois].

La deuxième partie contient la description d'un logiciel interactif de manipulation de graphes et d'ordres.

" ... the number of theorem in graph theory that were conjonctured as the result of mathematical doodling - the construction, manipulation, and examination of hand-drawn graphs - must surely be extremely large. Yet there is a danger in making conjectures from hand-drawn figures; small graph may well be atypical in their properties, while the drawing and manipulation of large graphs is more difficult and more prone to error. Hence a very useful adjunct to a graph-theorist's tool-kit is a computer program that will display graphs on a screen and permit their manipulation in various ways." [Read]

A notre connaissnce, la plus ancienne réalisation de ce type de logiciel est le "Graph theory interactive system" de WOLFBERG [Wolf].

Suivant les travaux de GUIDO sur CABRI (CAhier de BRouillon Informatisé), nous desirons offrir un poste de travail sur les graphes pour des chercheurs en théorie des graphes.

CABRI est une bonne approche du problème, mais reste d'un emploi malaisé.

Nous avons donc étudiés de nouveau le problème en nous attachant à décrire une meilleure interface utilisateur. Nous nous sommes inspirés des travaux sur les logiciels interactif existant, comme ceux développés chez XEROX, au PALO-ALTO RESEARCH CENTER [Smit] chez NIEVERGELT [Bere].

Une première ébauche de ce programme, écrit en C sur une machine UNIX† (SM90), a été réalisé au cours du mois de mai 1985.

références

[Andr] ANDRE J., MENU J., MUELLER J.P., Un exemple pédagogique pour Prolog, T.S.I., Vol.3, N°5, 1984, pp.361-363.

[Barb] BARBERYE G., JOUBERT T., MARTIN M., MOUFFRON M., PAUL E., Manuel OASIS (version printemps 83), note technique CNET NT/PAA/CLC/LSC/959, Mai 1983.

[Bere] BERETTA G., BURKHART H., FINK P., NIEVERGELT J., STELOVSKY J., SUGAYA H., VENTURA A., WEYDERT J., Xs1: An integrated intercative system and its kernel, Proc. 6th Int. Conf. Software Enginering, IEEE Computer Society, Tokyo, 1982, pp.340-349.

† UNIX est une marque déposée des laboratoires A. T. & T.

[Gang] GANGNET M., MICHELUCCI D., Un outil graphique interactif, Rapport de recherche n°84-12, Ecole des Mines de Saint-Etienne, Octobre 1984.

[Guid] Y. GUIDO, CABRI: un cahier de brouillon informatisé pour l'étude de la théorie des graphes, Thèse de 3ème cycle, Centre de recherche en informatique de Montpellier, Mars 1984.

[Grev] GREVISSE N., Le bon usage, 11ème édition, Duculot, Bruxelles, Belgique, 1982.

[Grof] GROFF, Manuel de référence version 2.5, Ecole des mines de Saint-Etienne, 1984.

[Mois] MOISSINAC J.C., Aide informatique à la réalisation de dessins animées, Thèse de docteur-ingénieur, Ecole des Mines de Saint-Etienne, 1984.

[Read] READ R.C,. Some application of computer in graph theory, dans Selected Topics in graph theory, edité par BEINEKE L.W. et WILSON R.J., Academic Press, 1978, pp.417-444.

[Sham] SHAMOS M.I., Computational geometry, Ph.D. Thesis, Dept. of comput. Sci., Yale University, New-Haven, Conn., 1978.

[Smit] SMITH D.C., IRBY C., KIMBALL R., VERPLANK B., HARSLEM E., Designing the Star User Interface, Byte Magazine, Vol.7, N°4, Avril 1982.

[Tall] TALLOT D., Sur la méthode de balayage du plan, rapport de recherche N°84-3, Ecole des Mines de Saint-Etienne, Avril 1984.

[Wolf] WOLFBERG M.S., An interactive graph theory system, Ph.D, Computer Science, University of Pennsylvania, 1969.

PREMIERE PARTIE

SUR L'ALGORITHMIQUE GEOMETRIQUE

INTRODUCTION

QUELQUES MOTS

SUR

LES TYPES ABSTRAITS

1. QUELQUES MOTS SUR LES TYPES ABSTRAITS

Citons DIJSKTRA

> "the amount of complexity that the human mind can cope with at any instant in time is considerably less that embodied in much of the software that one might wish to build".

Plusieurs techniques ont été proposées afin de réduire la complexité de la mise au point d'algorithmes et de programmes, pour permettre de vérifier l'exactitude de ces derniers et pour faciliter la maintenance des programmes. Les difficultés et les erreurs viennent particulièrement du nombre de paramètres à manipuler (dû souvent aux détails d'implémentation), et à l'existence de cas limites ne rentrant pas dans le schéma global.

Parmi ces techniques, citons:

- la décomposition en sous-problèmes et la programmation structurée introduit par DIJKSTRA et WIRTH [Wir1] [Dahl]
- l'abstraction [Hoa*] [Gut*]
- les langages de spécification [Abri] et l'emploi de langages comme MODULA [Wir2] ou ADA [Bern]
- la programmation logique avec des langages tel Prolog.

La définition de clauses en PROLOG est trés proche de la spécification du problème. On peut citer comme exemple la réalisation d'un programme de coupure des mots en fin de ligne d'un texte français [Andr], où les

clauses sont la traduction pratiquement directe des règles de coupure syllabique données dans le GREVISSE [Grev]. Cette méthode n'est pourtant pas sans risques, la définition de la négation d'une clause et l'exhaustivité dans la recherche des solution dans PROLOG ne "colle" pas bien avec les méthodes traditionnelles de spécification de problèmes.

L'emploi de ces techniques permet un exposé plus rigoureux, et donc plus clair des algorithmes. Par exemple, lorsque l'on trouve un invariant de boucle, on a vraiment compris ce que faisait la boucle en question, comment l'initialiser et comment l'arrêter.

Dans les annés 70, un axe de recherche fut le développement de la notion de type de données abstrait (in english: ADT).

Un type abtrait a deux aspects:

- Une spécification
- Une implémentation

La **spécification** définit une classe d'objets abstraits grâce aux opérations valides sur ces objets.

L'**implémentation** décrit l'équivalence avec d'autres types abstraits plus élémentaires et plus proches des types manipulés par les langages de programmation.

Plusieurs méthodes de spécification ont été mises au point:

- La spécification **opérationnelle**, qui décrit le type abstrait par "l'exemple", en termes de graphes, d'ensembles et de séquences. Par l'utilisation d'un langage semblable aux langages informatiques traditionnels, cette méthode est proche de l'informaticien.
- La spécification **définitionelle**, qui décrit le type abstrait par ces propriétés externes grâce à des axiomes, par exemple, dans une spécification algébrique (algèbre hétérogène [Birk]).

La première méthode est trop proche des langages informatiques traditionnels. On confond trop souvent types abstraits avec les "modules" de MODULA ou les "packages" de ADA. Décrire une pile informatique de façon abstraite nést pas spécifier les opérations "push" et "pop" par des manipulations d'indices d'un tableau, mais plutôt de la décrire comme un objet obéissant à la loi "LAST_IN, FIRST_OUT". L'utilisation des types abstraits facilite la preuve indépendamment de l'implémentation choisie et permet d'appréhender souvent toute une classe d'algorithmes en un seul formalisme.

L'utilisation de la spécification algébrique simplifie le développement des programmes par raffinements successifs, en repoussant les détails d'implémentation le plus tard possible, et en facilitant la vérification de l'exactitude de chaque étape du développement.

D'où la préférence dans la suite, pour une spécification du type algébrique plus complexe, mais plus souple, puissante et naturelle.

Une spécification algébrique comporte trois parties:

- une spécification **syntaxique.**
- une spécification **sémantique.**
- une spécification des **restrictions.**

La spécification syntaxique permet le même genre de verification sur les types des données et des resultats des opérations que dans les langages genre PASCAL.

La spécification sémantique consiste en un ensemble d'axiomes définissant les opérations par leurs relations réciproques.

La spécification des restrictions précise le domaine de validité pour l'application des opérateurs et les valeurs rendues en cas d'erreur. Il est, en général, indispensable de spécifier le traitement des erreurs, si le type abstrait doit être, un jour, implémenté (ce qui est sa raison d'être).

Si les axiomes ne donnent pas toute l'information nécessaire à la définition, la spécification du type est dite non complète. On ne peut alors déterminer le comportement d'un programme utilisant ces opérations.

Une spécification algébrique correcte devrait être prouvée complète et non contradictoire. Ces deux problèmes sont généralement indécidables.

L'emploi de certaines méthodes de construction de l'axiomatisation rend possible cette vérification. Certains logiciels, comme OASIS [Barb], développé en PROLOG au CNET/PARIS-A, dans un but de spécification d'auto-commutateurs pour le réseau téléphonique [Ugar], permettent la **preuve interactive de spécification.**

Deux sortes d'opérations permettent de construire une spécification:

- De **construction** d'un objet du type concerné
- **D'interrogation** sur un objet du type concerné et ayant comme résultat un objet d'un autre type; elles permettent de distinguer entre eux les objets de la classe

Toutes les opérations d'interrogation doivent être appliquées aux opérations de construction du type concerné, ce qui permet presque de vérifier que la spécification est complète [Gut1].

Dans la suite, les types abtraits seront décrits de la manière suivante inspirée par l'outil OASIS [Barb], actuellement disponible sur le MULTICS du CNET et bientôt sur SM 90:

Nom du type abstrait

Paramètre : type des objets utilisés dans la construction du type abstrait considéré.

Constante : objet utilisé comme origine pour la construction du type abstrait. (en général, l'ensemble vide).

Opérations : syntaxe des opérations avec les types sur lesquelles elles s'appliquent et les types des résultats.

Signification : ensemble d'axiomes explicitant les opérations.

Précondition : spécifie la validité de l'application de chaque opérateur et la valeur à retourner en cas d'erreur.

Certains types abstraits seront considérés comme prédéfinis:

- type abstrait entier
- type abstrait booléen
- type abstrait point

Les deux valeurs "erreur" et "indéfini" sont communes à tous les types abstraits. On supposera que toutes les opérations s'appliquant sur des données contenant l'une de ces deux valeurs rendra la valeur "erreur".

De même les types de base utilisés dans les langages de programmation peuvent être décrits par des spécifications algébriques. L'implémentation d'un type abstrait peut donc se décrire algébriquement.

L'implémentation d'une spécification algébrique est alors prouvée correcte par l'existence d'un homomorphisme entre l'algèbre de la spécification et l'algèbre de l'implémentation.

Les spécifications algébriques ne proposent que des fonctions sans effet de bord, ni modification des paramètres, ce qu'autorisent tous les langages de programmation classiques par souci d'efficacité. Ce genre de problèmes posés par la spécification algébrique sont discutés dans [Gut3].

La spécification algébrique donne toutes les informations nécessaires (c.à.d le comportement "extérieur") au programmeur pour utiliser les types décrits et pour prouver son programme. Elle permet à l'implémenteur d'ignorer l'utilisation qui en sera faite. Elle dresse une barrière entre utilisation et implémentation.

De plus, la spécification peut servir de documentation aux programmes en apportant les informations essentielles à leur compréhension, de manière plus efficace que les langages de programmation traditionels.

Les algorithmes proposés ont été développés par modifications successives (ce qui est une bonne méthode, si l'algorithme de base vérifie les propriétés essentielles du problème, c.à.d. il ne faut pas essayer, entre deux modifications, de construire un algorithme différent reposant sur des propriétés différentes). Il suffit alors de prouver les transformations mathématiques sur les spécifications associées aux modifications (bien souvent des changements de variables, des compositions de fonctions [Abri]). L'emploi de types abstraits permet d'ajouter de façon simple des opérations, donc de rajouter de nouvelles propriétés sans revenir sur des détails d'implémentation des structures de données.

La formalisation par les types abstraits permet, en outre, d'étudier uniquement les propriétés essentielles et indispensables de l'algorithme.

Elle est peut-être une bonne réponse à la question de portabilité des programmes, le choix d'une implémentation étant guidé par l'efficacité que l'on veut obtenir et par les possibilités offertes par le langage de programmation et par la machine hôte.

2. *REFERENCES*

[Barb] BARBERYE G., JOUBERT T., MARTIN M., MOUFFRON M., PAUL E., Manuel OASIS (version printemps 83), note technique CNET NT/PAA/CLC/LSC/959, Mai 1983.

[Bern] BERNARD L., Comment présenter simplement le traitement des types abstraits de données, T.S.I, Vol.3, n°5, 1984, p.319-326.

[Birk] BIRKHOFF G., LIPSON J.D., Heterogenous Algebras, J. Combinatorial Theory, Vol.8, 1970, pp.115-133.

[Dahl] O.J. DAHL, E.W. DIJKSTRA, C.A.R HOARE, Structured Programming, A.P.I.C. Studies in Data Processing, n°8, Academic press inc., 1972.

[Gut1] GUTTAG J., HORNING J.J., The Algebraic Specification Of Abstract Data Types, Acta Informatica, Vol.10, 1978, pp.27-52.

[Gut2] GUTTAG J.,Abstract Data Type and the development of data structures, C. of ACM, Vol.20, n°6, June 1977, pp.396-404.

[Gut3] GUTTAG J., HOROWITZ E., MUSSER D.R., Some Extensions to algebraic specifications, Proc. ACM Conf. on Language Design for Reliable Software, Mars 1977, pp.63-67.

[Hoa1] HOARE C.A.R., An axiomatic basis for computer programming, C. of ACM, Vol.12, n°10, October 1964, pp.576-583.

[Hoa2] HOARE C.A.R., Proof of correctness of data representation, Acta Informatica 1 (1972), pp.271-281.

[UGAR] de UGARTE G., Spécification formelle et validation de logiciels: l'exemple de la gestion des lignes d'abonné, Note Technique NT/PAA/CLC/LSC/1111 octobre 1983.

[Wir1] WIRTH N., Program development by stepwise refinement, C. of ACM, Vol.14, n°4, April 1971, pp.221-227.

[Wir2] WIRTH N., MODULA a Language for Modular Programming, Software-Practice and Experience, Vol.7, 1977, pp.3-35.

CHAPITRE 1

ALGORITHMES DE CALCUL D'ENVELOPPES CONVEXES

1. INTRODUCTION

Le problème de la recherche de **l'enveloppe convexe** de N points, se rencontre dans de nombreuses disciplines telles le graphique par ordinateur, l'automatique, la reconnaissance des formes, la recherche opérationelle, l'analyse numérique, l'analyse de données....

Rappellons que l'enveloppe convexe d'un ensemble S de N points dans un espace R^n peut se définir comme l'intersection de toutes les régions convexes contenant cet ensemble de points. Notons la C.

Les données du problème sont les **coordonnées** des N points de l'ensemble S dans un repère orthonormé.

Les **sommets, côtés et faces** de l'enveloppe convexe formeront le résultat. De plus, pour le problème dans le plan, les points de l'enveloppe convexe seront donnés dans l'ordre trigonométrique.

De nombreux papiers ont été écrits sur ce sujet. On peut citer le travail de pionnier de GRAHAM [Grah] en 1972.

Le principe de l'algorithme proposé par GRAHAM est le suivant :

- la recherche d'un point intérieur à C en temps constant
- puis le tri en O(NlogN) sur les points de S en fonction de leurs coordonnées polaires. Celles-ci sont calculées en fonction du point intérieur trouvé précédemment

- puis enfin le parcours de la liste ainsi obtenue afin de sélectionner les points de l'enveloppe convexe en O(N)

Ce parcours consiste grossièrement en l'exploration des triplets de points a,b,c consécutifs. On élimine b si l'angle formé par abc est concave.

Cet algorithme est donc en O(NlogN).

Et, un an après, la publication de l'algorithme de JARVIS [Jarv]. Il consiste en la recherche d'un point de C, puis dans le parcours de C en recherchant le point de S ayant le plus petit angle polaire par rapport au dernier point de C trouvé.

Cet algorithme est en O(|C|N).

Nous nous devons de signaler les travaux de SHAMOS ([Sha1],[Sha2]), premier recueil important sur la géométrie algorithmique, où l'on trouve une étude de la complexité du problème et un nouvel algorithme basé sur le principe "Divide and Conquer" en O(NlogN).

Enfin le papier de PREPARATA ET HONG [Prep] en 1977. L'algorithme présenté repose aussi sur le principe "Divide and Conquer". L'ensemble de points est séparé en deux domaines convexes d'intersection vide grâce à un tri préalable sur une des coordonnées. Cet algorithme est aussi en O(NlogN).

Dans la même année, W.F.EDDY propose dans [Eddy], un algorithme inspiré de la méthode de tri QUICKSORT [Sedg].

Dans [Yao], paru en 1981, est prouvé la validité de la borne inférieure pour un modèle tolérant les fonctions quadratiques. Cette borne est Ω(NlogN). AVIS [Avis] propose en 1982 une autre démonstration de cette borne en utilisant un arbre linéaire de décision (i.e. en chaque sommet de l'arbre, on peut évaluer une fonction linéaire de la donnée).

2. ENVELOPPES CONVEXES DANS LE PLAN.

On présente ci-après un formalisme généralisant des algorithmes déjà connus comme celui de EDDY ou de JARVIS. Il permet de calculer l'enveloppe convexe C, triée dans l'ordre trigonométrique, de N points dans le plan en O(N|C|).

Le coeur de cet algorithme est la fonction que nous avons appellée "convex", qui est décrite dans le paragraphe suivant.

2.1. LA FONCTION CONVEX

On définit une fonction, appelée **convex**, qui a pour données:

- Un segment s d'extrémités A et B, orienté de A vers B.
- Un ensemble P constitué de points p strictement "au-dessus"† du segment AB.

Elle donne comme résultat une **ligne polygonale** L joignant A à B. Cette ligne polygonale ajoutée au segment AB forme l'enveloppe convexe de P U {A,B}. Cette ligne polygonale est décrite de A vers B.

▸**Remarque** On peut noter que le calcul de convex en dimension un revient au calcul du maximum d'un ensemble de n nombres.

Cette fonction est basée sur le paradigme algorithmique "Divide and Conquer".

Pour calculer L, joignant A à B, on détermine un sous-ensemble de points de L, ce qui permet de trouver une ligne polygonale L' sans boucle joignant A à B.

Chaque segment s_i de cette ligne L' permet de déterminer un sous-ensemble P_i formé par les points de P au-dessus de s_i.

Récursivement, la fonction convex est appliquée sur chaque segment s_i de L' et le sous-ensemble P_i afin d'obtenir la ligne polygonale L_i (voir figure 2.1.1).

La simple concaténation des L_i, permet d'obtenir L.

On peut interpréter aussi l'algorithme comme l'élaboration **d'approximations** successives par les lignes polygonales L' de l'enveloppe cherchée, le nombre de points situés à l'extérieur diminuant à chaque niveau de récursivité.

On va définir plus précisement les objets manipulés par l'algorithme.

Un point est défini par :

```
x(point) → réel;
y(point) → réel;
```

---

† "au-dessus" est défini par le demi-plan positif grâce à l'orientation du segment AB.

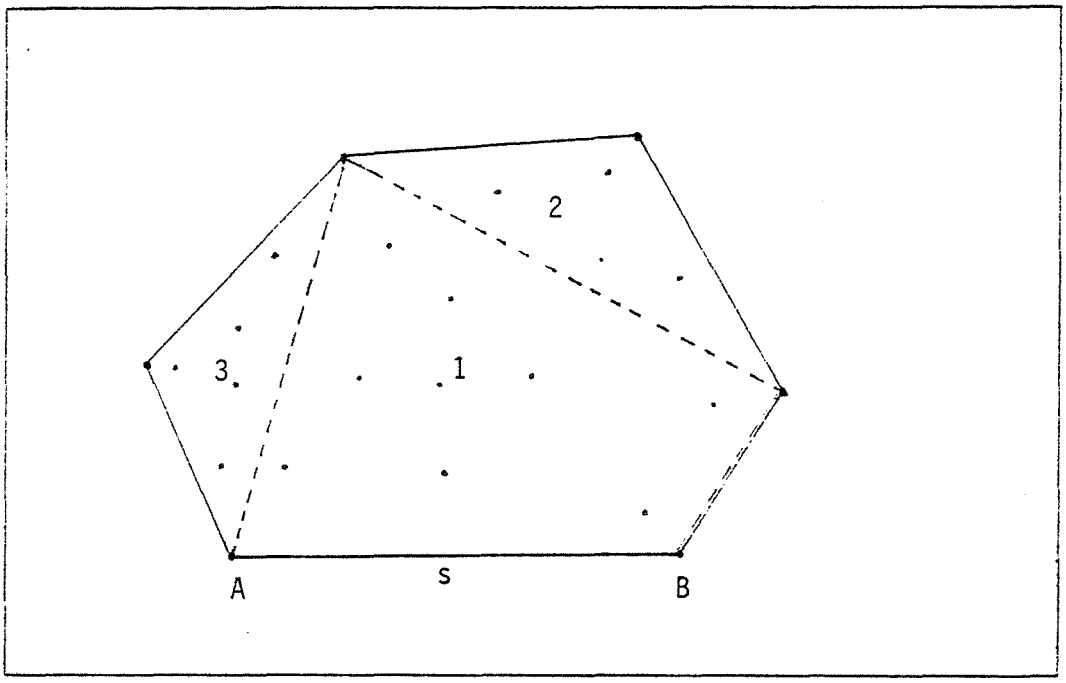


Figure 2.1.1

Un segment est défini par :

```
origine(segment) → point;
fin(segment) → point;
```

Un ensemble de points P est décrit par le type abstrait **P_structure**.

P_structure(point)

constante : P_structure_vide : P_structure;

opérations:

insérer(P_structure, point) → P_structure;
élément(P_structure) → point + {indéfini};
vide(P_structure) → booléen;
au_dessus(P_structure,segment) → P_structure;

définitions:

P: P_structure; p,p1,p2: point; s: segment;

élément(insérer(P,p)) → p;
vide(P_structure_vide) → vrai;
vide(insérer(P,p)) → faux;
au_dessus(P_structure_vide,s)→P_structure_vide;
au_dessus(insérer(P,p),s)
→ si distance(p,s)>0
alors insérer(au_dessus(P,s),p)
sinon au_dessus(P,s);

préconditions:

P=P_structure_vide => élément(P)→indéfini;

▸**Remarque** Cette P_structure suppose l'existence d'une fonction "distance(p,A,B)", où p,A et B sont des points, rendant un réel représentant la distance entre p et la droite passant par A et B et dirigée de A vers B.

On définit une structure de liste:

```
Liste(élem)

 constante : Liste_vide(élem) : Liste(élem);

 opérations:

  insérer(Liste(élem),élem) → Liste(élem);
  sup_premier(Liste(élem)) → Liste(élem);
  premier(Liste(élem)) → élem + {indéfini};
  concaténer(Liste(élem),Liste(élem)) → Liste(élem);
  vide(Liste(élem)) → booléen;

 définitions:

     L,L1,L2: Liste(élem);
     s: élem;

  sup_prem(Liste_vide(élem)) → Liste_vide(élem);
  sup_prem(insérer(L,s))
                  → si vide(L)
                  alors Liste_vide(élem)
                  sinon sup_premier(L);
  concaténer(L,Liste_vide(élem)) → L;
  concaténer(L1,insérer(L2,s))
                  → insérer(concaténer(L1,L2),s);
  premier(insérer(L,s))
                  → si vide(L)
                  alors s
                  sinon premier(L);
  vide(Liste_vide(élem)) → vrai;
  vide(insérer(L,s)) → faux;

 préconditions:

  L=Liste_vide(élem) => premier(L)→indéfini;
```

Une ligne polygonale sera décrite par une "Liste(segment)".

points_extrem(P_structure,p_1,p_2)→Liste(segment) fournit la ligne polygonale L', orientée de p_1 à p_2 formée par un sous-ensemble de points de la ligne recherchée L. Nous verrons dans le paragraphe suivant comment la réaliser.

diviser(P_structure,Liste(segment))→Liste(P_structure) construit à partir d'une P_structure P, une liste de P_structures P_i, chaque P_i étant formée du sous-ensemble de points de P au-dessus du ième segment de la liste de segments.

```
fonction convex(P: P_structure; AB :segment)
                    : Liste(segment);

s : segment;
P1 : P-structure;
LP : Liste(P_structure);
L' : Liste(segment);

début
si vide(P) alors
     convex ← insérer(Liste_vide(segment),AB);
     (* AB appartient à L *)
sinon
     début
     L' ← points_extrem(P,origine(AB),fin(AB));
     LP ← diviser(P,L');
     convex ← Liste_vide(segment);
     tant que non vide(L') faire
       début
       s ← premier(L');
       L' ← sup_premier(L');
       P1 ← premier(LP);
       LP ← sup_premier(LP);
       convex ← concaténer(L,convex(P1,s));
       fin;
     fin;
fin.
```

Algorithme 2.1.1

Nous utilisons la propriété suivante bien connue

Propriété 2.1.1

Soit P un ensemble de N points du plan.

Soit $p_1,\ldots,p_k$, k points de l'enveloppe convexe de P.

Alors les points situés à l'intérieur (strictement) du polygone $(p_1,\ldots,p_k)$ n'appartiennent pas à l'enveloppe convexe de P.

Et, si P_i est un sous-ensemble de points situé à l'extérieur de ce polygone et strictement au dessus du segment $p_i p_{(i+1)\bmod(k)}$ alors $\cap P_i P_j = \emptyset$, $1 \leqslant i \leqslant k$, $1 \leqslant j \leqslant k$.

Théorème 2.1.1

L'algorithme 2.1.1 calcule la fonction convex.

Démonstration :

L'algorithme 2.1.1 se termine car il trouve au moins un nouveau point de la ligne polygonale à chaque appel (en effet, dans les appels récursifs de convex, P1 ne contiendra plus ce point).

La démonstration se fait par induction sur le nombre de points contenus dans la P_structure P.

Si P est vide, alors AB est bien la ligne polygonale cherchée.

Si P n'est pas vide, "diviser" donne une liste de P_structures. Chacune de ces P_structures P_i est formées par l'ensemble des points au_dessus de chaque segment de L'.

Etudions la boucle "tant que".

L'invariant de cette boucle sera :

- à l'étape i, on traite le segment s de L' et P1, points au-dessus de s. L représente le morceau de la ligne polygonale cherché entre A et l'extrémité de s.

Cet invariant est vérifié lors de l'initialisation de la boucle.

Par hypothèse d'induction, convex avec comme données le segment s et la P_structure P1 ($|P1|<|P|$, donc l'hypothèse peut s'appliquer) donne, une ligne polygonale L1 entre l'origine et l'extrémité de s.

L1 est bien le morceau de la ligne cherchée entre l'origine et l'extrémité de s (propriété 2.1.1).

Par concaténation de L avec L1, on obtient la ligne cherchée entre A et l'extrémité de s et orientée de A vers l'extrémité de s.

Le segment s est alors supprimé de L'.

La boucle se termine sur le traitement du segment s finissant en B, L est donc bien la ligne cherchée entre A et B et orientée de A vers B. □

2.2. OPERATION "POINTS_EXTREM"

La définition de l'opération "points_extrem" influence fortement le comportement de l'algorithme.

Rappelons la définition de cette opération.

Elle fournit une ligne polygonale L' joignant A à B, dont les points sont choisis parmi ceux de l'enveloppe convexe.

Au moins deux solutions existent pour trouver un point de L':

- Choisir le point p le plus éloigné du segment AB parmi les points de la P_structure. Cela peut être réalisé en O(|P|).

- Choisir le point p de la P_structure formant le plus grand angle (p,A,B). La complexité est en O(|P|).

Dans les deux cas, L' sera formé par la liste de segments (Ap,pB).

La première solution permet d'obtenir un algorithme proche de celui de EDDY (voir figure 2.2.2).

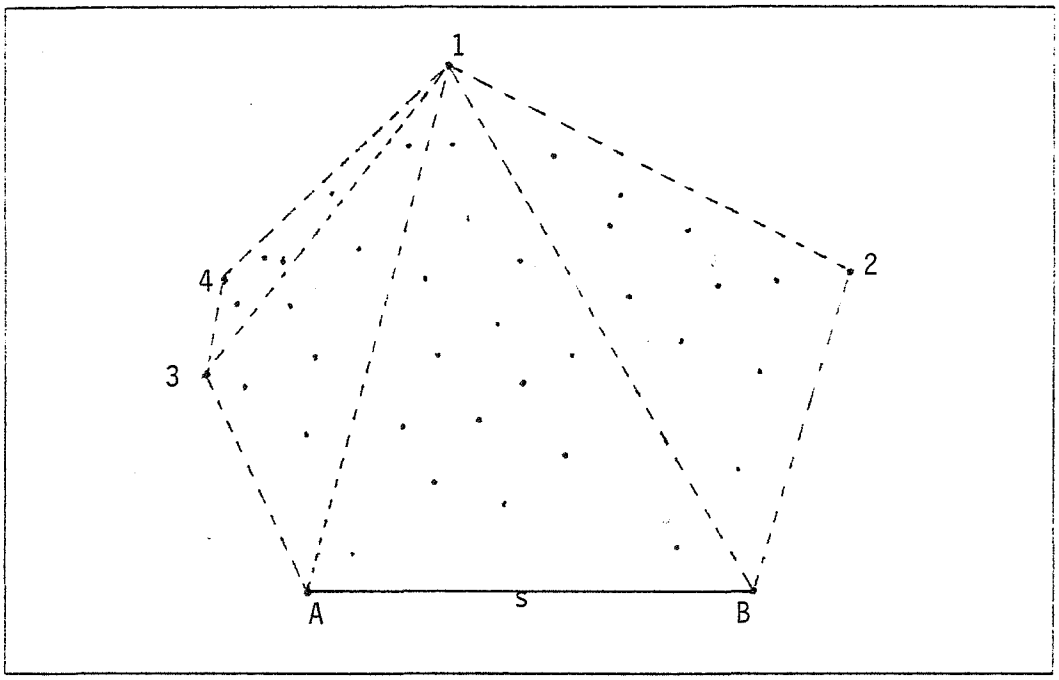


Figure 2.2.2

La deuxième solution aboutit à un algorithme semblable à celui de JARVIS (voir figure 2.2.3).

Cette solution permet une simplification de l'algorithme, en remarquant que la P_structure au-dessus de Ap est vide.

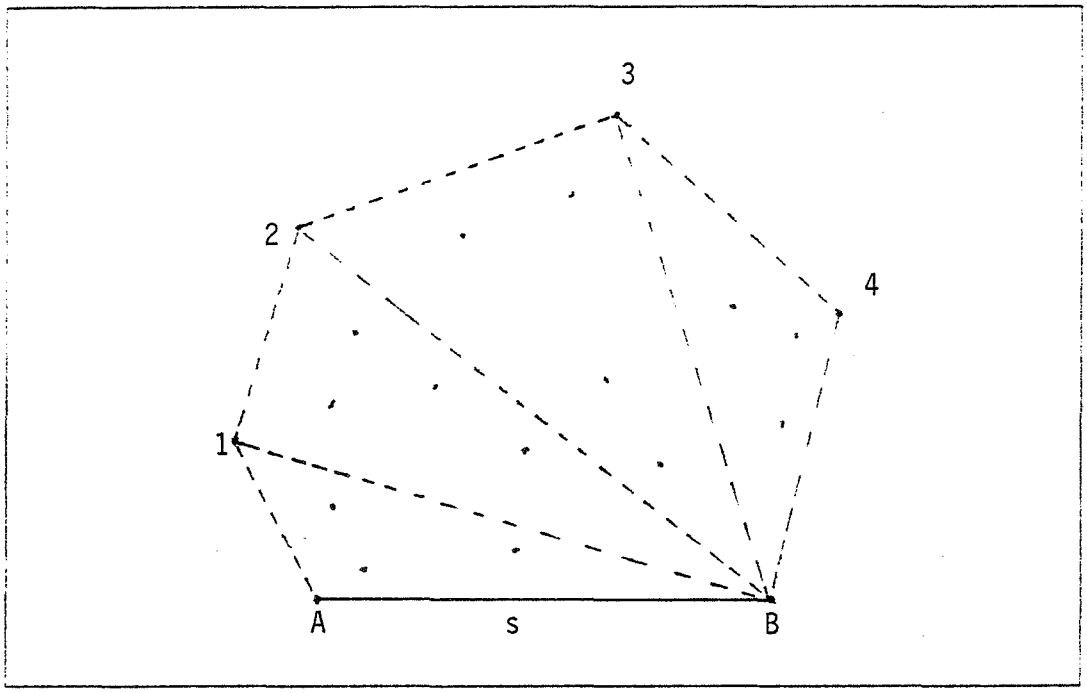


Figure 2.2.3

D'autres fonctions points_extrem pourraient être envisagées, aboutissant à des algorithmes différents.

Une fonction points_extrem permettant d'obtenir trois points pourrait être construite en associant les deux solutions précédentes. Le polygone ainsi formé aura une plus grande surface que le triangle précédemment obtenu avec un seul point, et on peut espérer que le nombre de points éliminés à chaque étape soit plus grand.

Une étude - à la manière de l'analyse fine de QUICKSORT faite par Sedgewick [Sedg] - du comportement d'un algorithme ayant une fonction points_extrem s'adaptant à la répartition des points dans le plan est en cours.

En conclusion, le formalisme proposé permet d'englober deux algorithmes différents ainsi que leurs généralisations.

2.3. ANALYSE EN COMPLEXITE.

Les lignes polygonales sont représentées par des listes de segments. Les insertions et les concaténations se faisant en fin de listes se font en O(1).

L'opération diviser fournit k sommets appartenant à l'enveloppe convexe à chaque niveau de récursivité. Dans les deux implémentations proposées k=2, pour notre analyse nous supposerons k constant.

A chaque niveau de récursivité, l'opération diviser est en O(k|P|).

Les N points sont stockés dans un tableau; chaque P_structure sera représentée par un sous-tableau connexe, c.à.d par deux indices de début et de fin.

L'opération diviser sur une P_structure P par rapport aux k segments de L' et l'obtention des k P_structures peut se faire en O(k|P|), en utilisant une version généralisée de l'algorithme du drapeau hollandais de DIJKSTRA (voir annexe 1).

Hormis les k appels récursifs, la procédure convex est en O(k|P|).

Comme on trouve k nouveaux points de L à chaque niveau de récursivité, la fonction est en O(|P||L|).

Bien sûr, tous les points de P peuvent appartenir à l'enveloppe convexe et dans ce cas, la complexité est alors de $O(|P|^2)$.

Cette complexité est atteinte pour un cas comme celui de la figure 2.3.4, où tous les points appartiennent à l'enveloppe convexe, et où le partitionnement de P donne deux P_structures, P1 et P2, telles que |P1|=0 et |P2|=|P|-1.

2.4. ALGORITHME D'ENVELOPPE CONVEXE

Cette fonction "convex" permet d'obtenir l'algorithme 2.4.2 pour calculer l'enveloppe convexe.

A partir de deux points de l'enveloppe convexe, on applique la fonction "convex" sur le segment AB et l'ensemble de point au-dessus de AB, et sur le segment BA et l'ensemble de points au-dessous. L'enveloppe convexe est obtenu par concaténation, dans le bon ordre, des deux lignes polygonales trouvées.

La détermination de deux points de l'enveloppe convexe peut être faite d'au moins 2 façons:

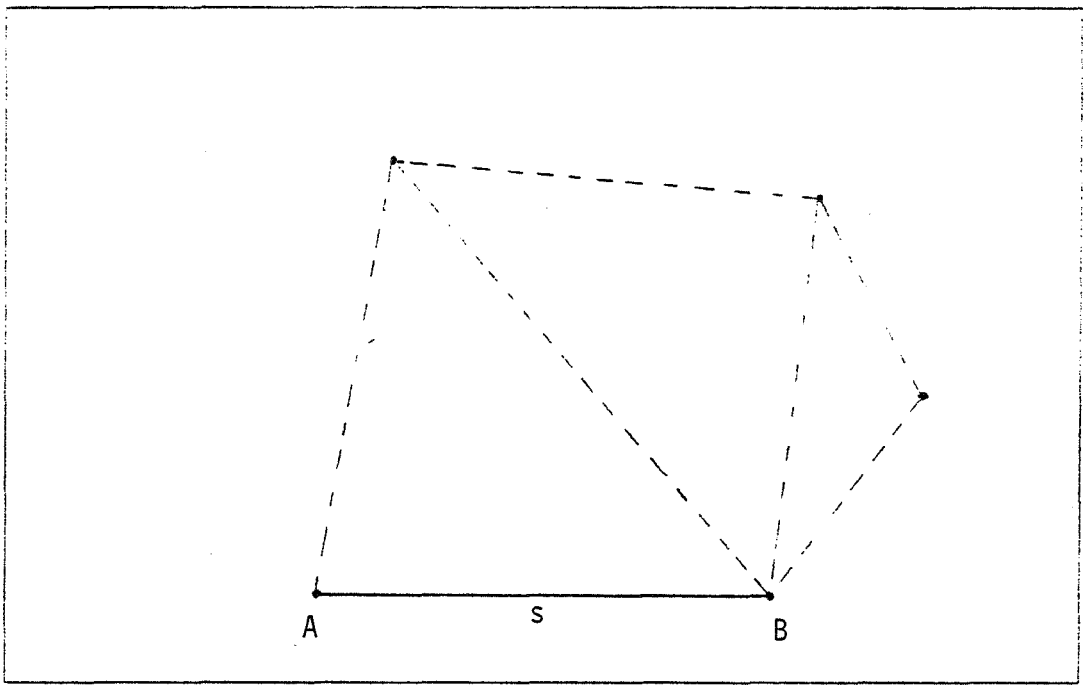


Figure 2.3.4

```
P,P1,P2 : P_structures;
C : L_structure;
A,B : points;

début

(* initialisation de la P_structure *)
P ← P_structure_vide;
pour chague point p faire
   P ← insérer(P,p);

(* calcul de l'enveloppe convexe *)
A,B ← 2 points de l'enveloppe convexe de P;
P1 ← au_dessus(P,AB);
P2 ← au_dessus(P,BA);
C ← concaténer(Convex(P1,AB), Convex(P2,BA));

(* obtention de l'enveloppe convexe par la *)
(* L_structure C et l'opération élément *)

fin.
```

Algorithme 2.4.2

- calcul d'une enveloppe convexe en 1D de la projection des N points sur une droite quelconque.
- calcul par l'emploi d'un algorithme d'enveloppe convexe en 2D déjà connu comme celui de JARVIS [Jarv].

La démonstration de cet algorithme est évidente et emploie les mêmes arguments que la précédente.

Analyse en complexité

L'initialisation de la P_structure se fait en O(N).

Le calcul des deux points extrêmaux peut être fait en O(N). La concaténation est en O(1), et chaque appel de fonction coûte respectivement O(|P1||L1|) et O(|P2||L2|).

Or |P1|+|P2|=N et |L1|+|L2|=|C|, donc la complexité de l'algorithme est donc en **O(|C|N)**.

Pour analyser les performances en moyenne, on a besoin d'une estimation de |C|, pour certaines distributions dans une région fermée. Des résultats de ce type peuvent être trouvés en géométrie stochastique (voir [Sant],p.22).Nous ne rappelerons que quelques résultats sur |C|.

Soit $_2K$ une région fermée avec une frontière ∂K de classe C^2, et de courbure $\kappa(s)$ et H_n l'enveloppe convexe de n points donnés au hasard dans K, alors, pour le nombre v de points de H_n, on a :

$$E(v) \backsim \Gamma(5/3)\ (2/3)^{1/3}\ (F^{-1/3}\ Q_{1/3})\ n^{1/3}$$

où F est la surface de K et

$$Q_m = \int_0^L \kappa^m ds$$

Si K est un polygone convexe avec r sommets, on a :

$$E(v) = 2/3r(\log n + C) + 2/3\log(F^{-r}\ \Pi_1^r f_i) + O(1)$$

où C=0,5772.. est la constante d'Euler, F la surface de K et f_i la surface du triangle $A_{i-1}\ A_i\ A_{i+1}$, avec A_i notant les sommets ordonnés de K.

On a donc :

- $|C| = O(N^{1/2})$ pour N points choisis uniformément dans un cercle.

- $|C| = O(\log N)$ pour N points choisis uniformément dans un polygone convexe.

Ce qui donne pour complexité en moyenne:

- $O(N^{4/3})$ pour calculer l'enveloppe de N points répartis uniformément dans un cercle.

- $O(N \log N)$ pour calculer l'enveloppe de N points répartis uniformément dans un polygone.

Ces complexités sont à comparer avec celles en $O(N \log N)$, dans le plus mauvais cas, des algorithmes de GRAHAM et de PREPARATA/HONG.

2.5. CAS DE DEGENERESCENCE.

On dira que l'enveloppe convexe est dégénérée si au moins trois points de cette enveloppe sont alignés.

Si l'opérateur "au-dessus" rend les points strictement au-dessus du segment, le resultat sera une enveloppe convexe comportant uniquement les points nécessaires.

Au contraire, si la comparaison est faite au sens large, l'enveloppe pourra contenir des points alignés.

3. REFERENCES

[Avis] AVIS D., On the complexity of finding the convex hull of a set of points, Discrete Applied Mathematics, n°4, 1982, p.81-86.

[Chan] CHAND D.R., KAPUR S.S., An algorithm for convex polytopes, J. ACM, Vol.17, n°7, Janvier 1970, p.78-86.

[Dijk] DIJKSTRA E.W., A Discipline of programming, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

[Eddy] EDDY W.F., A new algorithm for the convex hull of a planar set, ACM Trans. Math. Software, Vol.3, n°4, Decembre 1977.

[Grah] GRAHAM R.L., An efficient algorithm for determining the convex hull of a finite planar set, Inform. Proc. Lett. 1, 1972, p.132-133.

[Jarv] JARVIS R.A., On the identification of the convex hull of a finite set of point in the plane, Inform. Proc. Lett. 2, 1973, p.18-21.

[Prep] PREPARATA F.P., HONG S.J., Convex hull of finite sets of points in two and three dimensions, Comm. ACM, Vol.20, n°2, Feb. 1977, p.87-93.

[Sant] SANTALO L.A., Integral geometry and geometric probability, Encyclopedia of mathematics and its applications, G.C. Rota editor, Dep. of Math., MIT, Cambridge, Massachusetts, 1976.

[Sedg] SEDGEWICK R., The Analysis of Quicksort Programs, Acta Informatica, vol.7, 1977, p.327-355.

[Sha1] SHAMOS M.I., Problems in computational geometry, Dep. Comptr. Sci., Yale U., New Haven, Conn., May 1975.

[Sha2] SHAMOS M.I., Computational geometry, Ph.D, Yale University, 1978.

[Yao] YAO A.C., A lower bound to finding convex hulls, JACM, vol.28, n°4, october 1981, pp.780-787.

CHAPITRE 2

BALAYAGE DU PLAN

1. INTRODUCTION

Le calcul des intersections de segments dans le plan et l'obtention d'une structure de graphe planaire à partir d'un dessin formé de segments sont des problèmes fréquemment rencontrés lors de l'élaboration d'algorithmes sur le traitement et la synthèse de dessins [Gang] [Mois].

Un algorithme naïf consiste à tester tous les couples de segments et à calculer leurs intersections. Cet algorithme est évidemment en $\theta(n^2)$ pour une donnée de n segments.

Dans ce chapitre, nous allons définir un ordre sur les segments du plan, et expliciter l'algorithme utilisé pour calculer celui-ci. Cet ordre est utilisé pour obtenir un algorithme de calcul d'intersections en O((n+k)logn) ou k est le nombre d'intersections.

Cet algorithme, du type "ratissage", est basé sur le balayage du plan, "de droite à gauche", par une ligne de référence, sautant d'un point au suivant. D'une position à une autre de la ligne de référence, des structures de données bien choisies permettent de maintenir sans trop de calculs un ordre partiel local sur les segments.

Nous reviendrons sur le principe de cette méthode dans l'annexe 1 sur le paradigme du ratissage.

Shamos et Hoey ont introduit la technique du balayage du plan dans [Sha2] afin d'obtenir un algorithme optimal pour tester l'existence d'un point d'intersection parmi un ensemble de segments du plan.

Bentley et Ottman proposent dans [Bent] une modification de l'algorithme précédent pour calculer tous les points d'intersection.

Enfin, Nievergelt et Preparata exposent dans [Nier], deux nouvelles applications de la méthode de balayage du plan, pour la décomposition d'un polygone croisé et pour l'intersection de 2 cartes planaires convexes.

Cette méthode a été utilisé par Gangnet et Michelucci [Gang], pour la réalisation d'un logiciel de saisie de plans architecturaux et d'esquisses à partir du dessin à main levée sur un table à numériser.

Le paragraphe 2 de ce papier se propose de donner un exposé clair (hum!) et propre (re-hum!) de la méthode de balayage du plan, valable dans tous les cas de figures (c.à.d. segments colinéaires, intersection formée de plus de 2 segments,...), et mettant en évidence les structures de données et les opérateurs associés suffisants pour implémenter l'algorithme.

Le problème de l'intersection de segments est développé dans le paragraphe 3. On rappelle les bornes inférieures obtenues par Shamos. On remarque, au passage, que l'on peut alors obtenir la structure de graphe planaire sous-jacente à cet ensemble de segments. On étudie aussi un problème simplifié, où on ne considère plus que des ensembles de segments ayant uniquement un nombre fixé de directions possibles. Cet algorithme est optimal.

Le paragraphe 4 expose une propriété nécessaire à l'extension du balayage à d'autres objets géométriques.

2. ORDRE SUR LE PLAN ET PRINCIPE DU BALAYAGE

Nous allons donc définir un ordre sur les segments du plan, et expliciter l'algorithme utilisé pour le calculer. Cet ordre et cet algorithme seront utilisés dans le paragraphe suivant pour résoudre le problème du calcul des intersections de segments.

2.1. DEFINITION DE L'ORDRE SUR LES SEGMENTS

Définition 2.1.1

Considérons S un ensemble de n segments distincts du plan.

Considérons une droite B du plan appelée droite de référence ou axe de référence.

Nous la choisirons de telle sorte qu'il n'existe aucun segment de S parallèle à B.

Orientons B. (i.e. nous définissons un ordre total τ_B sur les points de B)

Cet ordre s'étend à S de la manière suivante:

$s', s'' \in S$

$s' <\tau_B s'' <=> s' \cap B <\tau_B s'' \cap B$

ou $s' \cap B = s'' \cap b$
et $<B, s'> < <B, s''>$

ou $s' \cap B = s'' \cap B$
et $<B, s'> = <B, s''>$
et longueur de s' < longueur de s''

▸**Remarque** $<B, s>$ est l'angle formé par B et s.

Sauf cas particulier, τ_B est une relation d'ordre partiel sur S.

En effet, il existe des segments incomparables, dans la figure 2.1.1, on a $A_1 <\tau_B A_2$ et $A_1 <\tau_B A_0$, mais A_3 n'est pas comparable à A_1, ni à A_2 pour B mais on a $A_3 <\tau_C A_2$ et $A_4 <\tau_C A_3$

Définition 2.1.2

Faisons passer en chaque extrémité ou intersection de S une droite parallèle à B. Nous avons ainsi au plus 2n+k droites B_i parallèles à B, s'il y a k intersections.

Ces 2n+k droites divisent le plan en au plus, 2n+k+1 morceaux que nous appelerons régions.

Propriété 2.1.1

τ_B est invariant pour toute translation de B à l'intérieur d'une région (**bord droit non compris**)

On peut facilement en conclure que pour une direction B et pour un ensemble de n segments se coupant en k points, il y a, au plus, 2n+k+1 régions où τ_B soit différent.

Toutes les droites B_i sont orientées de la même manière que B. Chacune de ces droites sépare le plan en deux demi-plans, à "gauche" et à "droite".

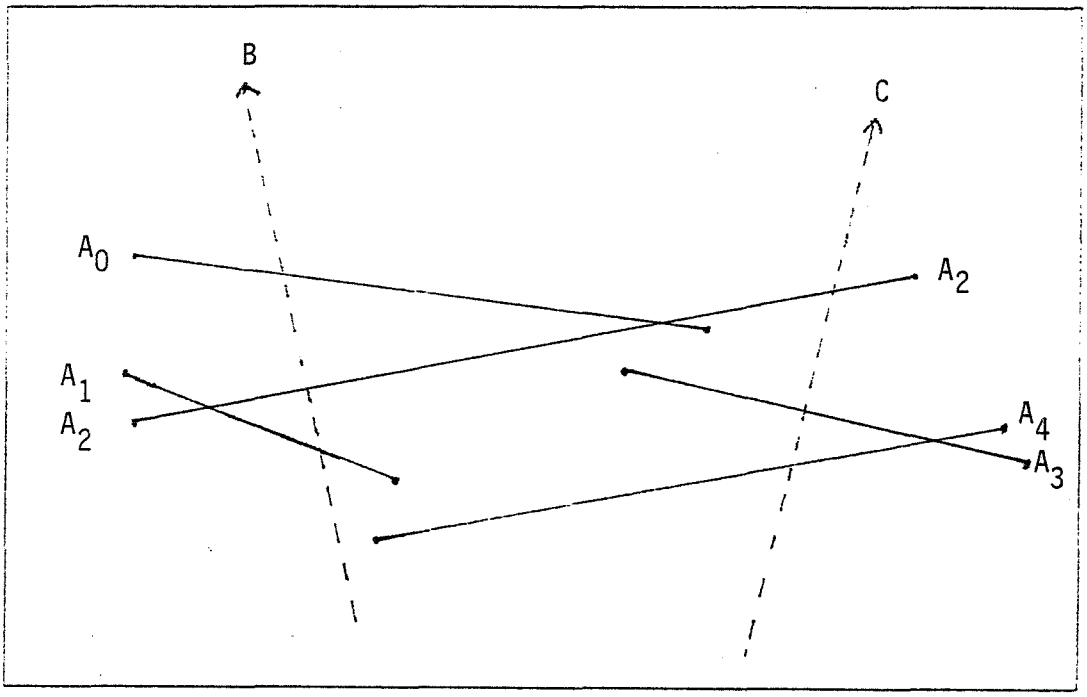


Figure 2.1.1

On numérote de façon naturelle, les droites B_i de gauche à droite. On fait de même pour les régions (voir figure 2.1.2).

2.2. STRUCTURES DE DONNEES ET ALGORITHME

Nous allons décrire un algorithme qui calcule successivement les 2n+k+1 relations d'ordre τ_B pour un axe de référence B.

Grossièrement, l'algorithme "balaye" le plan, calcule τ_B sur chaque région successivement en remarquant qu'au passage de la frontière entre deux régions, il est facile et peu coûteux de transformer l'ordre sur S:

1 - On supprime tous les segments finissant sur la ligne de référence considérée.

2 - On insére tous les segments commençant sur cette ligne, à leur place dans τ_B.

3 - Tous les segments se coupant en un point sur cette droite voient leur places s'inverser dans l'ordre.

On peut considérer l'ensemble des points, origines, extrémités, intersections et segments comme un **graphe planaire G=(V,E)** où V est constitué de toutes les **extrémités et intersections** et E par les **morceaux de seg-**

ments

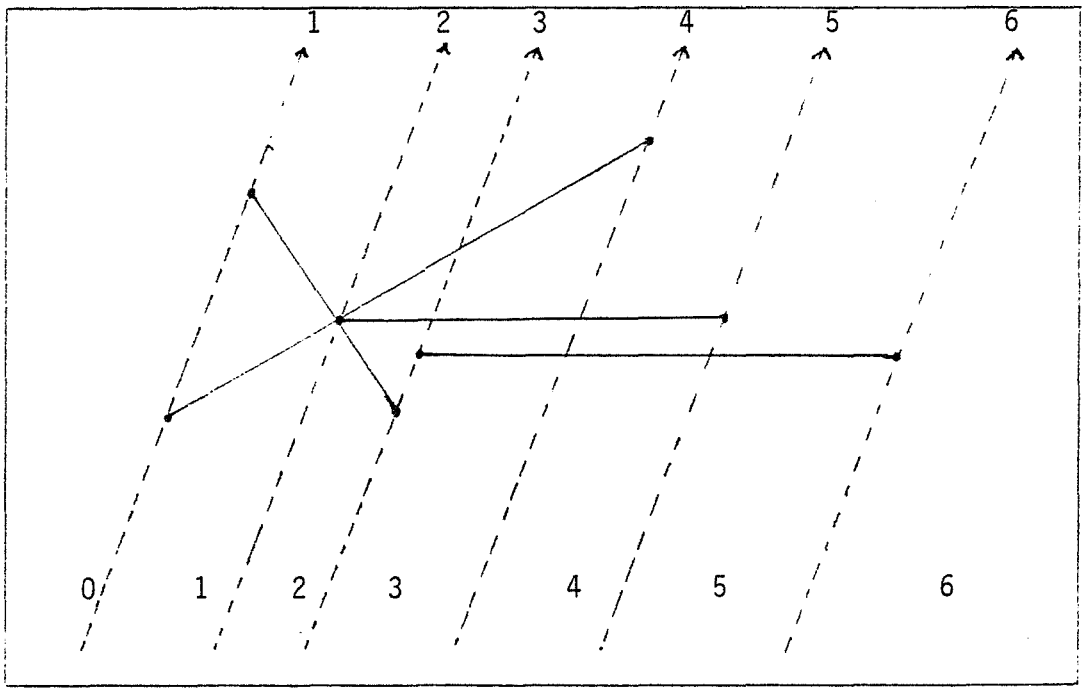


Figure 2.1.2

compris entre deux points.

Chaque point sera **éclaté** en autant de **sommets** que son degré dans le graphe G.

Le choix de cette représentation, différente de celle utilisé dans [Bent] et [Nier], permet de simplifier l'algorithme, comme nous pourrons le voir plus loin, en supprimant le traitement particulier associé aux points d'intersection.

Les lignes de références seront décrites par les opérations suivantes:

```
axe_de_référence←nouvel_axe_de_référence();

axe_de_référence←passant_par(coordonnées);
```

La première opération fournit un axe de référence passant par $-\infty$.

Les **segments** seront décrits par trois opérations:

```
segment←nouveau_segment(coordonnées,coordonnées);

coordonnées←origine(segment);

coordonnées←extrémité(segment);
```

A chaque sommet, sera associé ses coordonnées, son type, et le segment auquel il est lié.

```
sommet ← nouveau_sommet(coordonnées,type,segment);

coordonnées ← coord(sommet);

type ← typ(sommet);

segment ← seg(sommet);
```

Le type d'un point, "origine" ou "extrémité", sera défini par:

> Soit un segment s et soient p_1, p_2 les deux extrémités de ce segment
>
> On a $P_1 \in B_i$ et $p_2 \in B_j$, B_i et B_j axes de référence, et de plus $i \neq j$ (par hypothèse)
>
> p_1 est l'origine de s et p_2 est l'extrémité de s si $i<j$ et vice-versa.

Ainsi, l'algorithme maintient dynamiquement une structure ordonnée qui contient uniquement la partie significative de τ_B, i.e. les segments comparables. Cette structure sera appelée **Y_structure** (ou segment_structure).

Les opérations nécessaires à cette Y-structure sont les deux opérations de construction, insérer et supprimer un segment, ceci afin de maintenir l'ordre sur les segments.

Nous rajouterons l'opération min(Y_structure), qui permettra de distinguer entre eux les éléments de la classe Y_structure; on pourra grâce à elle obtenir l'ordre des segments de la Y_structure.

Une deuxième structure, appellée **X_structure** (ou point_structure) contiendra tous les sommets définis ci-dessus, sommets pour lesquels une modification de l'ordre des segments est nécessaire (propriété 2.1.1).

Les sommets seront classés dans la X_structure par l'ordre induit par la ligne de balayage.

Soient p_1 et p_2, 2 sommets

Il existe 2 droites B_i et B_j telles que
$p_1 \in B_i$ et $p_2 \in B_j$

$p_1 < p_2$ <=> i<j
ou si i=j alors type(p_1)=extrémité
et type(p_2)=origine
ou si type(p_1)=type(p_2)
alors segment(p_1) < segment(p_2)

La X_structure sera munie des opérations d'insertion d'un point afin de l'initialiser. Les opérations d'obtention et de suppression du point minimum permettrons la simulation du balayage. On devra aussi pouvoir tester si cette structure est vide.

▸**Remarque** On suppose tous les segments distincts (ce qui se vérifie facilement en O(n)).

Le maintien de la Y_structure suppose que la X_structure contienne tous les sommets correspondant à des points d'intersection.

On expose une propriété nécessaire à l'existence d'intersections de segments en relation avec l'ordre défini en 2.2.1. sur les segments.

Propriété 2.2.2

Si les segments $s_1, s_2, \ldots\ldots, s_l$ se coupent en un point L, alors il existe une droite B_i de référence pour laquelle ces segments sont contigus.

Démonstration

On note $S=\{s_1, s_2, \ldots\ldots, s_l\}$ (voir fig.2.2.3).

Supposons qu'il existe un segment qui brise la contiguïté de S pour la droite de référence passant en P. Ce segment ne passe pas par L. Il est situé entre s_i et s_j, donc il coupe obligatoirement soit s_i, soit s_j.

Donc P n'est pas le point le plus proche de L, d'où contradiction. □

Cette propriété exprime une condition nécessaire à l'existence d'un point d'intersection.

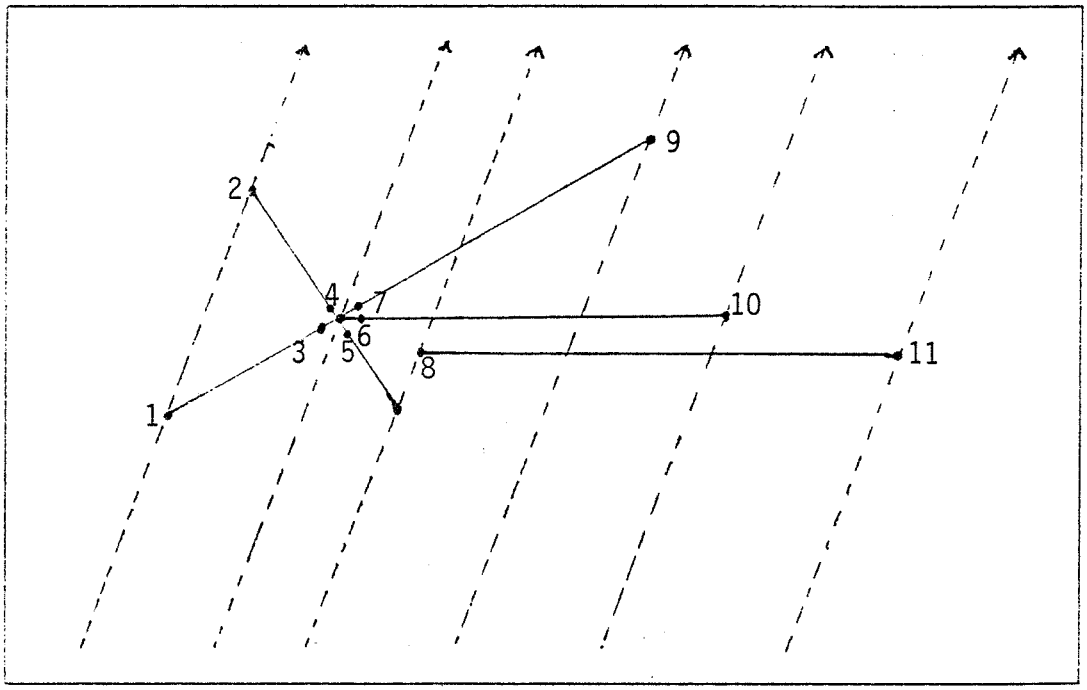


Figure 2.2.3

Pour trouver les points d'intersection, l'algorithme devra tester toutes les nouvelles adjacences entre segments apparaissant lors des transformations de l'ordre pour chaque région.

Pour tester les adjacences entre segments, on aura besoin de deux opérations de plus sur la Y_structure, **au_dessus** et **en_dessous**, donnant respectivement le segment après et avant un segment donné dans l'ordre de la X_structure.

Un problème: l'algorithme peut trouver plusieurs fois la même intersection entre les 2 mêmes segments (voir fig.2.2.4).

Si ces points sont insérés sans précaution, l'ordre ne pourra plus être correctement maintenu dans la Y_structure.

D'où le besoin de rajouter à la X_structure la possibilité de **vérifier l'existence d'un sommet** avant de l'insérer.

On trouvera plus loin la description des deux structures à l'aide de types abstraits.

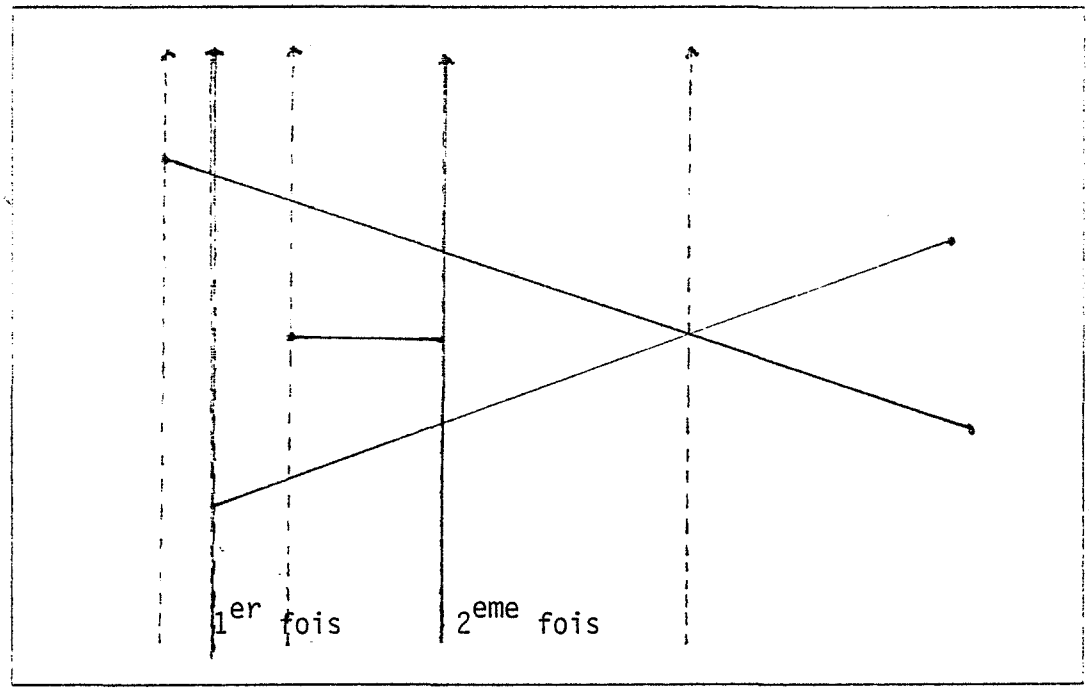


Figure 2.2.4

Les points d'intersection seront insérés dans la X_structure au fur et à mesure de leur découverte, cela pour les points appartenant à une ligne de balayage supérieure à la ligne de balayage courante (ceux qui sont inférieurs n'interviendrons plus pour le maintien de l'ordre des segments).

La mise à jour de la X_structure sera faite de la manière suivante:

Soit i, les coordonnées du point d'intersection,
s_1 et s_2, les deux segments concernés
On définit:

```
X_structure ←
     mise_à_jour(X_structure,coordonnées,segment,segment);

mise_à_jour(S,i,s1,s2) →
    si i ≠ extrémité(s1)
        alors p ← nouveau_sommet(i,origine,s1)
              si non existe(S,p) alors insérer(S,p)
              p ← nouveau_sommet(i,extrèmité,s1)
              si non existe(S,p) alors insérer(S,p)
    si i ≠ extrémité(s2)
        alors p ← nouveau_sommet(i,origine,s2)
              si non existe(S,p) alors insérer(S,p)
              p ← nouveau_sommet(i,extrèmité,s2)
              si non existe(S,p) alors insérer(S,p)
```

▸**Remarque** Mise_à_jour est conçu de telle manière que l'on n'insère pas un sommet confondu avec une extrémité d'un segment.

On dispose aussi d'une fonction de calcul d'intersection:

intersection(segment,segment) → point + {indéfini};

▸**Remarque** La procédure d'intersection n'est pas décrite. On peut trouver une description de toutes les bonnes méthodes dans [Luca]. En particulier on peut utiliser des conditions nécessaires à partir de la notion de boite englobante, ceci accélère grandement le temps de calcul.

Y_structure

paramètres : segment
axe de référence
constante : Y_structure_vide : Y_structure

opérations :
insérer(Y_structure,segment,axe)→Y_structure
supprimer(Y_structure,segment,axe)→Y_structure
min(Y_structure,axe)→segment + {indéfini}
max(Y_structure,axe)→segment + {indéfini}
vide(Y_structure)→booléen
au_dessus(Y_structure,segment,axe)→segment
+ {indéfini}
au_dessus(Y_structure,segment,axe)→segment
+ {indéfini}

signification :
S=Y_structure;
s, s_1=segment;
B=axe de référence;

supprimer(Y_structure_vide,s)→Y_structure_vide;
supprimer(insérer(S,s_1,B),s,B)
→ si s=s_1 alors S
sinon insérer(supprimer(S,s,B),s1,B);

vide(insérer(S,s,B))→faux;
vide(Y-structure_vide)→vrai;

min(insérer(S,s,B),B)
→ si vide(S)
alors s
sinon si min(S,B) > s alors s
sinon min(S,B);

max(insérer(S,s,B),B)
→ si vide(S)
alors s
sinon si max(S,B) < s alors s
sinon max(S,B);

au_dessus(S,s,B)
→ si min (S,B)<τ_B s
alors au_dessus(supprimer(S,min(S,B),B),s,B)
sinon min(supprimer(S,s,B),B);

en_dessous(S,s,B)
→ si max (S,B)>τ_B s
alors en_dessous(supprimer(S,max(S,B),B),s,B)
sinon max(supprimer(S,s,B),B);

précondition :
S=Y_structure_vide => min(S)→indéfini;
S=Y_structure_vide => max(S)→indéfini;

X_structure

paramètre : sommet

constante : X_structure_vide : X_structure

opérations :
insérer(X_structure,sommet)→X_structure
suppmin(X_structure,sommet)→X_structure
min(X_structure)→sommet + {indéfini}
vide(X_structure)→booléen
existe(X_structure,sommet)→booléen

signification :
S=X_structure;
p_1, p_2=sommet;

suppmin(X_structure_vide)→X_structure;
suppmin(insérer(S,p_1))
→si p_1=min(insérer(S,p_1)) alors S
sinon suppmin(S);

min(insérer(S,p_1))
→si vide(S) alors p_1
sinon si min(S) > p_1 alors p_1
sinon min(S);

vide(insérer(S,p_1))→faux;
vide(X_structure_vide)→vrai;

existe(X_structure_vide,p_1) → faux;
existe(insérer(S,p_1),p_2)
→ si p_1=p_2 alors vrai
sinon existe(S,p_2);

précondition :
S=X_structure_vide => min(S)→indéfini;

Théorème 2.2.1

L'algorithme 2.2.1 calcule bien les τ_{Bi}.

Démonstration

L'algorithme se termine quand la X_structure est vide. Chaque tour de boucle supprime un sommet, et le nombre de sommets est fini.

La démonstration se fait par récurrence.

```
Données : n segments différents

p: sommet;
B,BP : axes de référence;
s1,s2,maxs,mins : segments;

(* INITIALISATION *)

Y_structure ← Y_structure_vide;
X_structure ← X_structure_vide;
pour chaque segment s faire
   début
   p ← nouveau_sommet(origine(s),origine,s);
   X_structure ← insérer(X_structure,p);
   p ← nouveau_sommet(extrémité(s),extrèmité,s);
   X_structure ← insérer(X_structure,p);
   fin;

(* BALAYAGE DU PLAN *)

B ← nouvel_axe_de_référence;
tant que non vide(X_structure) faire
   début
   p ← min(X_structure);
   si passant_par(coord(p)) ≠ B
      alors
       début
       (*  On a ici l'ordre sur les segments   *)
       (*  comparables pour la région compris  *)
       (*  entre BP et B, B non compris  que   *)
       (*  l'on peut obtenir grace aux         *)
       (*  opérations min et vide sur la       *)
       (*  Y_structure                         *)
       BP ← B;
       B ← passant_par(coord(p));
       fin;
   suivant typ(p) faire
    origine : début
              Y_structure ←
                 insérer(Y_structure,seg(p),B);
              s1 ←
                 au-dessus(Y_structure,seg(p),B);
              s2 ←
                 en-dessous(Y_structure,seg(p),B);
              si s1 ≠ indéfini
               alors
                début
                i ← intersection(s1,seg(p));
                si i non indéfini et i > p
                alors
                X_structure ←
                mise_à_jour(X_structure,i,s1,seg(p));
                fin;
              si s2 ≠ indéfini
               alors
```

```
                début
                i ← intersection(s2,seg(p));
                si i non indéfini et i > p
                alors
                X_structure ←
                mise_à_jour(X_structure,i,s2,seg(p));
                fin;
              fin;
    extrémité : début
            s1 ← au-dessus(Y_structure,seg(p),B);
            s2 ← en-dessous(Y_structure,seg(p),B);
            si s1 ≠ indéfini et s2 ≠ indéfini
               alors
                début
                i ← intersection(s1,s2);
                si i non indéfini et i > p
                alors
                X_structure ←
                mise_à_jour(X_structure,i,s1,s2);
                fin;
            Y_structure ←
               supprimer(Y_structure,seg(p),BP);
            fin;
   X_structure ← suppmin(X_structure);
   fin;
fin;
```

Algorithme 2.2.1

L'ordre est correct pour la première région, celle-ci ne contenant aucun segment et la Y_structure étant initialisée à vide.

hypothèse

L'ordre est correct dans la Y_structure pour la région i-1, bord gauche compris et bord droit non compris.

On va en déduire l'ordre pour la région i.

On change de droite de référence.

Soit B_i la droite séparant les deux régions.

Soit P l'ensemble des sommets de type origine et extrémité appartenant à B_i.

Les sommets de P sont traités l'un après l'autre, suivant leur type, d'abord les sommets de type extrémité, puis les sommets de type origine et dans l'ordre de la ligne de balayage de "bas en haut" (voir la définition du paragraphe 2.2.1).

2 cas sont à considérer:

1.p est un sommet de type origine (fig.2.2.5)

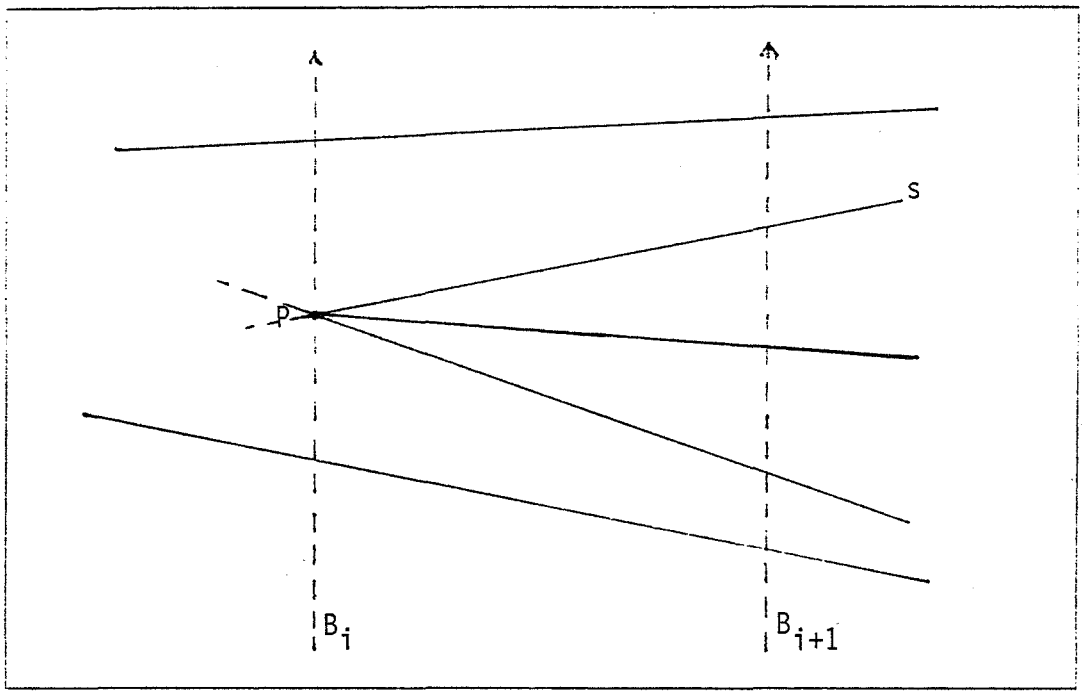


Figure 2.2.5

Soit s, le segment commencant en p.

Ce segment n'appartient pas à la Y-structure. Si p est un point d'intersection, alors p a été traité avant comme point de type extrémité, donc s a été supprimé, sinon on n'a jamais rencontré s. On insère ce segment dans la Y_structure pour l'axe de référence B_i. Il sera dans le bon ordre pour la région i.

Il y a un changement d'adjacence. On teste les intersections entre s_0 et s_1 et entre s_1 et s_2, et on insére ces points si ils existent, sont supérieurs à la ligne de balayage courante (c.à.d à p) et n'ont pas déjà été insérés.

2.p est un sommet de type extrémité (fig.2.2.6)

Soit s, le segment finissant en p.

On supprime ce segment dans la Y_structure pour l'axe de référence B_{i-1} (c.a.d l'axe précédent B_i), l'ordre sur B_i étant différent de celui de la région i-1 (voir la définition des régions). On peut remarquer que tous les segments finissant sur B_i coupent obligatoirement B_{i-1} et qu'ils appartiennent donc à la Y_structure (toujours par hypothèse de récurrence).

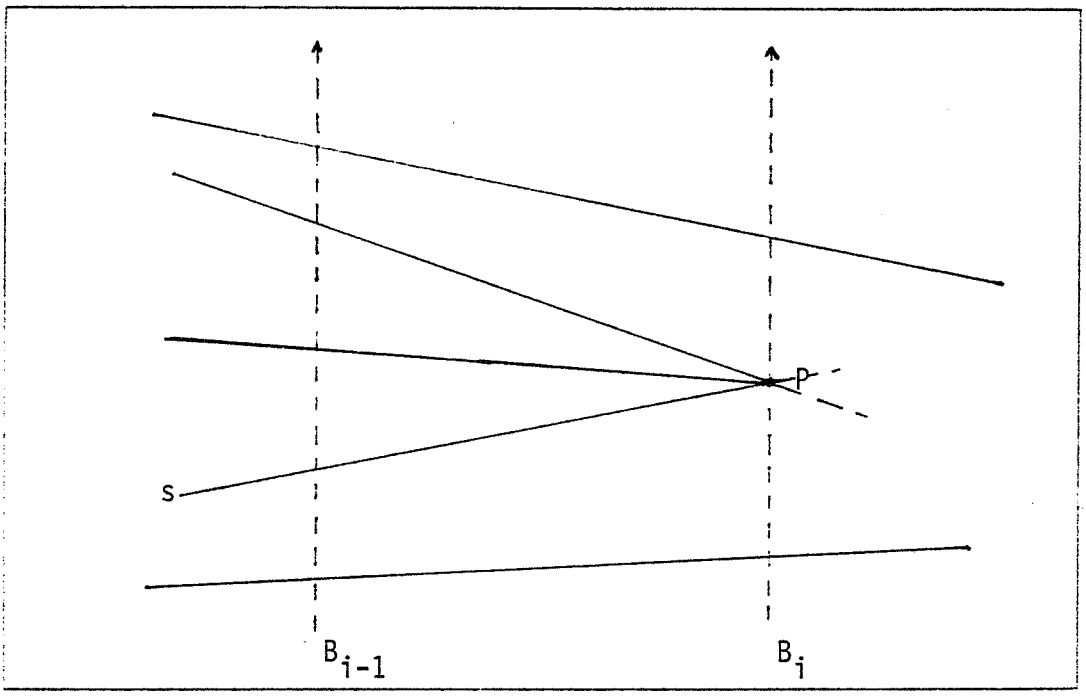


Figure 2.2.6

Il y a un changement d'adjacence. On teste l'intersection entre s_0 et s_2 et on insére ce point si il existe, est supérieur à la ligne de balayage courante (c.à.d à p) et n'a pas déjà été inséré.

Remarque B_1 ne contient que des points de type origine, on n'utilisera donc jamais B_0, qui est à $-\infty$.

Il est évident que lorsque P a été entièrement traité, on connait l'ordre sur la région i.

Soit B_{i+1} la droite de référence suivante, existe-t-il un point d'intersection non trouvé entre B_i et B_{i+1} ?.

Soit p un tel point.

Il est l'intersection de l segments $s_1, s_2, \ldots, s_l$. Tous ces segments coupent B_i et sont adjacents dans l'ordre τ_B, sinon il existerait:

- soit un point extrémité entre B_i et p ce qui est impossible
- soit un point d'intersection auquel cas, on applique une seconde fois le raisonnement sur ce point.

Donc p aurait été découvert et inséré dans la Y_structure.

D'où la contradiction

▸**Remarque** L'existence de segments colinéaires d'intersection non nulle ne modifie en rien l'algorithme car elle ne modifie pas l'ordre de la Y_structure pendant le balayage.

Donc à la fin de l'algorithme, pour chaque région, on connaît l'ordre τ_B qui lui est associé. □

Analyse en complexité

Il est évident que la Y_structure et la X_structure peuvent être implémentées sous, au moins, trois formes:

- liste ordonnée.
- liste ordonnée associé à un arbre équilibré en hauteur (AVL).
- tas ou queue de priorité (heap).

On connaît les résultats suivants, sur le nombre de comparaisons entre éléments, pour un ensemble de n éléments (voir tableau 2.2.1).

tableau 2.2.1

	liste triée	liste+AVL	tas
Insertion	O(n)	O(log(n))	O(log(n))
Suppression	O(n)	O(log(n))	O(n)
Min	O(1)	O(log(n))	O(1)
Vide	O(1)	O(1)	O(1)
Suppmin	O(1)	O(log(n))	O(log(n))
au_dessus	O(1)	O(1)	O(n)
En_dessous	O(1)	O(1)	O(n)
Existence	O(n)	O(log(n))	O(n)

Les opérations "au_dessus" et "en_dessous" sur la Y_structure peuvent se faire en un temps constant que ce soit pour une liste triée ou pour un arbre AVL dont toutes les feuilles sont doublement chaînées dans l'ordre (c.à.d une structure où une liste triée est associée à un AVL).

Soit n le nombre de segments et k le nombre d'intersections

Lemme

Le nombre total de sommets contenus dans la X_structure est au maximum $2(3(2n+k)-7) = 14n +6k -14$.

Démonstration

En effet, on peut considérer l'ensemble des n segments comme un graphe planaire à 2n+k sommets et m arêtes.

$$\Sigma_i d_i = 2m <= 2(3(2n+k)-7) \text{ par la formule d'Euler}$$

où d_i est le degré du sommet i et chaque point est éclaté dans la X-structure autant de fois que son degré.

Il y a donc, au plus 12n+6k-14 sommets dans la X_structure. □

La Y_structure contient au maximum n segments.

La première boucle du programme est formée de 2n insertions dans la X_structure.

La deuxième boucle est exécutée 12n+6k-14 fois. Elle génère, au maximum, donc:

-12n+6k-14 vide
-12n+6k-14 min
-12n+6k-14 suppmin sur la X_structure

-12n+6k-14 insertion/supression
-12n+6k-14 au_dessus
-12n+6k-14 en_dessous
-2(12n+6k-14) mise_à_jour sur la Y_structure

-2(12n+6k-14) intersection entre deux segments

L'opération de mise à jour comporte:

- 4 appels à existence dans la X_structure

- 4 appels à insérer dans la X_structure

Le calcul d'une intersection entre deux segments se fait en temps constant.

On en déduit la complexité en nombre de comparaisons entre segments:

X_structure et Y_structure sous forme de liste triée
$O((n+k)^2)$

X_structure sous forme de tas et
Y_structure sous forme d'AVL
$O((n+k)^2)$

X_structure et Y_structure sous forme d'AVL
$O((n+k)\log(n+k))$

On teste, au maximum, 2 intersections par tour de boucle. La complexité en nombre de calcul d'intersection est en $O(n+k)$.

3. INTERSECTIONS DE SEGMENTS DANS LE PLAN

Nous commencerons par étudier l'application de la méthode de balayage du plan à l'intersection de segments dans le plan et à l'obtention du graphe planaire sous-jacent à cet ensemble de segments.

Les données du problème sont n segments données par les coordonnées de leurs extrémités dans un repére orthonormé. Le re'sultat sera les coordonnées des k points d'intersection avec les segments associés.

3.1. BORNE INFERIEURE

Comme d'habitude, il y a deux types de problèmes, **existence et calcul.**

Soient n segments

P1: Existent-t-il au moins deux segments parmi n, ayant une intersection non vide ?
P2: Rechercher tous les points d'intersection de cet ensemble de segments.

Une borne inférieure sur le nombre de comparaisons pour P1, à partir du problème en dimension 1, est donnée dans [Sha1] p.123.

La technique employée pour établir cette borne inférieure est celle de la **T-réductibilité** entre problèmes.

On rappelle les définitions de Shamos[Sha1].
Soient deux problèmes A et B.

1. les données de A sont transformées en des données correctes pour B.
2. le problème B est résolu.
3. les résultats de B sont transformés en des des résultats corrects pour A.

Si les transformations 1 et 3 sont faites en O(T(n)), alors on dit que A est T(n)-réductible à B et on note:

$$A <<_{T(n)} B$$

Cette relation de préordre, qui est un cas particulier de la réduction polynômiale utilisée dans la théorie P-NP, semble être un outil intéressant pour une classification précise des problèmes polynômiaux, malgré quelques difficultés de notations et de mise en oeuvre. Sur ces difficultés, on peut consulter les travaux de HABIB, HAMROUN, JEGOU [Habi].

On a la propriété suivante:

Propriété 3.1.3

Si le problème B peut être résolu en T(n) et $A <<_{R(n)} B$, alors A peut être résolu en T(n)+O(R(n)).

Pour ce qui suit, afin de ne pas rentrer dans les détails de notations, nous admettrons que ce préordre permet de trans mettre aussi les bornes inférieures.

Nous avons alors le théorème suivant:

Théorème 3.1.2 [Shamos]

O(nlogn) comparaisons sont nécessaires et suffisantes pour déterminer si n intervalles sont disjoints, si uniquement le calcul de fonctions polynômiales des données est autorisé.

La preuve repose sur l'équivalence entre P1 et le problème de l'unicité d'un élément dans une famille de cardinal n.

$$\text{unicité d'un élément} <<_n \text{intervalles disjoints}$$

D'autre part, il est évident que

$$P1 <<_n P2$$

De plus, si on note P_1^i le problème P1 pour un ensemble de segments en dimension i, il est trivial de démontrer (par projection) que

$$P_1^i <<_n P_1^{i+1}$$

de même

$$P_2^i <<_n P_2^{i+1}$$

On en déduit une borne inférieure pour P1 sur un ensemble de n segments dans le plan

$$\Omega(n\log n)$$

et pour P2

$$\Omega(n\log n + k)$$

où k est le nombre d'intersections trouvé.

3.2. CALCUL DES INTERSECTIONS

Shamos et Hoey montrent dans [Sha2] que le balayage du plan donne un algorithme optimal pour P1.

Bentley et Ottman proposent dans [Bent] une modification de l'algorithme précédent pour résoudre le problème P2 lorsqu'on suppose qu'en un point, au plus 2 segments se coupent. Cet algorithme est annoncé en $O((n+k)\log(n+k))$.

Brown dans [Brow], propose une amélioration pour la complexité en place mémoire.

Dans [Bent], il est recommandé d'utiliser un tas pour la X_structure. Malheureusement, l'opération "existence", qui est indispensable au bon fonctionnement de l'algorithme, demande alors $O(n)$ opérations élémentaires. Cela aboutit à un algorithme en $O((n+k)^2)$ contrairement a la complexité en $O((n+k)\log(n+k))$ annoncée dans ces articles.

L'algorithme 1 permet, sans modification, d'obtenir tous les points d'intersection, ceci pour tous les cas de figure (intersection en un point d'un nombre quelconque de segments, segments colinéaires,...).

Il permet, par conséquent, d'obtenir aussi la représentation de l'ensemble des segments sous forme de **graphe planaire** dont les sommets sont les extrémités et les intersections.

La complexité de cet algorithme est $O((n+k)\log(n))$.

L'algorithme n'est donc pas optimal pour la borne inférieure donnée dans le paragraphe 3.1.

Ceci provient de l'insertion dans la X_structure des points d'intersection trouvés. On peut, peut-être, se passer de faire cette opération. Le paragraphe suivant montre que l'on peut atteindre cette borne pour certains cas particuliers (ensemble de segments n'ayant que deux directions).

3.3. UN AUTRE PRINCIPE SANS CALCUL D'INTERSECTION

Ce paragraphe expose un autre algorithme de calcul d'intersections ne reposant pas sur le balayage du plan, mais sur les algorithmes de génération de segment, point par point, dans une "mémoire d'image".

Il est inspiré des algorithmes de tri sans comparaisons comme Radix-sorting.

De nombreux algorithmes de tracé de segment sur une grille existent: algorithme de Bresenham, de Stockton, de Lucas, de Earnshaw entre autres.

Le problème est de trouver les points les plus proches du segment à tracer. Arrivé en un point donné, ces algorithmes choisissent le point suivant parmi les huits voisins possibles. Ce point est alors "allumé" dans la mémoire d'image.

Trouver les points d'intersection consiste alors à tracer les segments les uns après les autres en testant pour chaque point choisi si celui-ci a déjà été allumé.

Une mémoire d'image munie de plusieurs plans permet en plus de mémoriser à quel segment appartient chaque point allumé et ainsi de trouver tous les points d'intersection et les segments concernés.

Cet algorithme calcule évidemment les points d'intersection avec une précision dépendante de la précision de la console de visualisation.

Si l'on choisit pour opération élémentaire le test d'un point de la grille d'affichage, la complexité est proportionnelle à la somme des longueurs de tous les segments. Elle n'est évidemment pas comparable à celle de l'algorithme 2.2.1, les opérations de base comptées n'étant plus les mêmes, en particulier la borne inférieure ne s'applique pas.

3.4. CAS PARTICULIER

De nombreuses applications n'ont à traiter que des ensembles de segments dont le nombre de directions est connu. Par exemple, pour la gestion d'un multifenêtrage ou de tableurs, les segments n'ont que deux directions: horizontale et verticale. De même, pour le traitement de plan d'architecte ou une application de C.A.O. en VLSI, il y a peu de directions différentes.

3.4.1. Ensemble de segments avec deux directions

On considère, dans la suite, un ensemble de segments ayant seulement **deux directions possibles.** Ces deux directions seront notées D_1 et D_2.

Une opération, **direction(segment)**, sera ajoutée à la définition des segments.

Le but est de trouver toutes les intersections entre les segments de direction D_1 et ceux de direction D_2.

L'axe de référence sera choisi de direction D_1 (on peut aussi choisir D_2, mais il faut bien prendre une décision). Grâce à ce choix, les points d'intersection trouvés seront situés sur la ligne de balayage courante et n'auront pas à être triés. L'algorithme pourra ainsi être optimal.

On définit plus précisément les X_structure et Y_structure employées par l'algorithme.

La Y_structure contiendra tous les segments (et seulement ceux-ci) de direction **D2**, coupés par la ligne de balayage et dans l'ordre décrit dans le paragraphe 2.

Les opérations **seg_sup** et **seg_inf** généralisent au_dessus et en_dessous, acceptant comme donnée un point appartenant à l'axe de référence plutôt qu'un segment.

```
seg_sup(Y_structure,point,axe)→segment

seg_sup(S,p,B)
    → si min(S,B)∩B <τ_Bi p
            alors seg_sup(supprimer(S,min(S,B),B),p,B)
            sinon min(S,B);

seg_inf(Y_structure,point,axe)→segment

seg_inf(S,p,B)
    → si max(S,B)∩B >τ_Bi p
            alors seg_inf(supprimer(S,max(S,B),B),p,B)
            sinon max(S,B);
```

La X_structure contient tous les points **origine et extrémité des segments de direction D_2**, ceci pour maintenir la Y_structure. Elle contiendra aussi les points **origine des segments de direction D_1**.

On définit maintenant l'ordre de ces points dans la X_structure.

Les droites de référence sont classées comme au paragraphe 1 de droite à gauche. On partitionne l'ensemble des points suivant leur appartenance à une droite de référence B_i ; et on classe ces partitions notées P_i dans le même ordre que B_i.

A l'intérieur de chaque sous-ensemble, les points seront classés par:

- d'abord les points origine des segments D_2
- puis les points origine des segments D_1
- enfin les points extrémité des segments D_2

A l'intérieur de ces classes, l'ordre est celui des points sur l'axe de référence, les cas d'égalité étant partagés par la longueur des segments.

La composition de ces ordres donnent l'ordre dans la X_structure.

L'algorithme travaillera suivant la méthode suivante. Lorsque la ligne de balayage rencontre un segment s de direction D_1, on extrait de la Y_structure tous les segments compris entre l'origine et l'extrémité de s, segments qui coupent obligatoirement le segment s (voir fig.3.4.1.7).

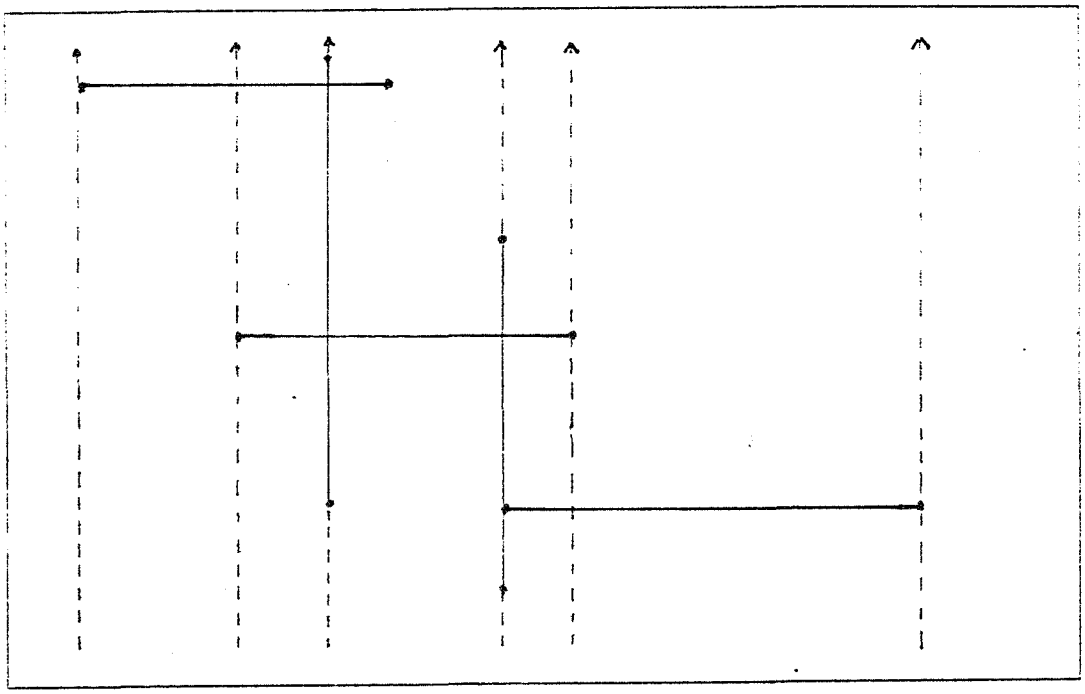


Figure 3.4.1.7

L'ordre défini pour les points est inspiré par la remarque suivante: pour trouver **tous** les points d'intersection, il faut d'abord avoir insérer tous les segments D_2 commencant sur la ligne de balayage, et n'avoir retirer aucuns segments D_2 finissant sur la ligne de balayage.

```
données: n1 segments différents dans la direction D1.
         n2 segments différents dans la direction D2.
résultats: tous les couples formés d'un segment D1 et
           d'un segment D2 se coupant.

s,segmin,segmax:segment;
p,pp:point;
B:axe de référence;

(* INITIALISATIONS *)

Y_structure←Y_structure_vide;
X_structure←X_structure_vide;
pour chaque segment s faire
   début
   suivant direction(s) faire
      D1: début
          p←nouveau_sommet
               (coord(origine(s)),origine,s);
          X_structure←inserer(X_structure,p)
          fin;
      D2: début
          p←nouveau_sommet
               (coord(origine(s)),origine,s);
          X_structure←inserer(X_structure,p);
          p←nouveau_sommet
               (coord(extrémité(s)),extrémité,s);
          X_structure←inserer(X_structure,p);
          fin
   fin

(* BALAYAGE DU PLAN *)

B ← nouvel_axe_de_référence;
tant que non vide(X_structure) faire
   début
   p←min(X_structure);
   si passant_par(coord(p)) ≠ B
      alors B ← passant_par(coord(p));
   suivant direction(seg(p)) faire
      D1: début
          pp←fin(seg(p));
          segmin←seg_sup(Y_structure,p,B);
          segmax←seg_inf(Y_structure,pp,B);
          s←segmin;
          i←intersection(s,segment(p));
          tant que s ≠ segmax faire
             début
             s←au_dessus(Y_structure,s,B);
             i←intersection(s,segment(p));
             fin
          fin;
      D2: si typ(p)=origine
            alors
             Y_structure←inserer(Y_structure,s,B)
```

```
            sinon
              Y_structure<-supprimer(Y_structure,s,B);
    X_structure←suppmin(X_structure);
    fin;
fin
```

Algorithme 3.4.1.2

Théorème 3.4.1.3

L'algorithme 3.4.1.2 trouve bien tous les points d'intersection entre les segments de directions D_1 et les segments de direction D_2.

Démonstration

La première boucle consiste en l'insertion d'un nombre fini de points dans la X_structure.

La deuxième et dernière boucle supprime à chaque passage un point dans la X_structure et finit quand celle-ci est vide.

Donc l'algorithme se termine toujours.

lemme

Lors du traitement d'un point origine d'un segment D_1, la Y_structure contient bien tous les segments D_2 coupés par l'axe de référence.

On peut remarquer que, de plus, les segments dont l'origine est sur l'axe de référence ont été insérés avant le traitement du point origine du segment D_1, et que ceux qui finissent sur l'axe de référence n'ont pas encore été supprimés, ce qui permet de trouver les points d'intersection confondus avec les extrémité de segments. □

Le traitement d'un point origine D_1 consiste à trouver tous les segments D_2, appartenant à la Y_structure, dont les intersections avec l'axe de référence sont compris entre l'origine et l'extrémité du segment D_1.

segmin et segmax prennent respectivement les valeurs des segments de la Y_structure immédiatement au-dessus de l'origine et immédiatement en-dessous de l'extrémité du segment D_1 (on ne traite pas les segments réduits à un point).

La variable s est initialisée avec la valeur du segment segmin.

Tous les segments sont trouvés par une boucle tant que. L'invariant de cette boucle est " tous le segments appartenant à la Y_structure compris entre segmin et s ont été trouvés".

La variable s prend la valeur du segment au dessus de s dans la Y_structure. Si s est égal à segmax, alors on a bien trouvé tous le segments compris entre segmin et segmax, sinon l'invariant est toujours vrai.

Le segment D_1 etant confondu avec l'axe de référence, on trouve bien toutes les intersections possibles. □

Analyse en complexité

Soit n_1 le nombre de segments de direction D_1.

Soit n_2 le nombre de segments de direction D_2.

Soit k le nombre d'intersections entre segments D_1 et D_2.

L'initialisation de la X_structure comporte:

$$n_1 + 2(n_2) \text{ insertions}$$

Le balayage du plan est réalisé par une boucle sur les $(n_1 + 2n_2)$ points de la X_structure.Il y aura donc:

n_2 insertions
n_2 suppressions
k au_dessus
n_1 seg_sup
n_1 seg_inf sur la Y_structure

$(n_1 + 2n_2)$ min
$(n_1 + 2n_2)$ suppmin sur la X_structure

k calcul d'intersection

Les opérations seg_sup et seg_inf de la Y_structure s'effectuent en O(log(n)) pour un AVL.

La X_structure contient $(n_1 + 2n_2)$ points et la Y_structure, au maximum, n_2 segments.

La complexité est donc:

$$O((n_1+2n_2)\log(n_1+2n_2) + 2(n_1+n_2)\log n_2 + k)$$

donc

$$O((n_1+n_2)\log(n_1+n_2) + k)$$

Le nombre d'intersections calculé est en $\theta(k)$.

Théorème 3.4.1.4

L'algorithme 3.4.1.2 est donc optimal pour le nombre de comparaisons et pour le nombre de calculs d'intersection.

3.4.2. Généralisation à un nombre constant de directions

On peut facilement étendre l'algorithme 3.4.1.2 à un nombre h de direction par (h-1) balayages.

Soit d_1, d_2, ..,d_h, les h directions de l'ensemble de segments.

Le ième balayage, fait avec un ligne de référence de direction d_i et sur les segments de direction d_i, d_{i+1}, .., d_h, permet de calculer les intersections entre les segments de direction d_i et les segments de direction d_{i+1}, d_{i+2},..., d_h.

Soit k_i, le nombre d'intersection entre les segments de direction d_j, $j \geq i$.

Soit n_i, le nombre de segments de direction d_i.

On a:

$\Sigma_1^h k_i = k$, où k est le nombre total d'intersections

$\Sigma_1^h n_i = n$, où n est le nombre total de segments

La complexité est alors en:

$$\Sigma_{i=1}^{h-1} ((n - \Sigma_{j=1}^{i-1} n_j) \log(n - \Sigma_{j=1}^{i-1} n_j)) + k$$

$$< (h-1)n\log n + k$$

4. EXTENSIONS DU BALAYAGE DU PLAN A D'AUTRES OBJETS GEOMETRIQUES

Il serait intéressant d'étendre le balayage du plan à d'autres objets que les segments.

Il semble que ces objets doivent avoir au moins la propriété essentielle suivante.

Propriété 4.4

Pour obtenir un ordre sur la ligne de balayage, l'intersection de l'objet avec la ligne de balayage doit être réduite à un point et un seul.

On peut appliquer ainsi les algorithmes décrits à des figures comportant des arcs de cercles. Au cour d'un prétraitement, ces arcs de cercle devront être divisés au niveau des points tangents à la direction de la droite de balayage, ceci afin de former des morceaux d'arcs de cercle ne pouvant couper la ligne de balayage qu'en un seul point.

On peut, en fait, appliquer cette méthode à de nombreuses courbes continues à condition de pouvoir calculer les points tangents à la doite de balayage.

5. CONCLUSIONS.

Les algorithmes sur des problèmes géométriques deviennent très complexes dès que l'on s'attache à traiter tous les cas limites étant susceptible de se produire.

L'utilisation des types abstraits permet une simplification de l'énoncé des algorithmes ainsi que de leur démonstration. En mettant en lumiere les propriétés fondamentales des structures de données employées, elle permet de choisir l'implémentation minimale la plus efficace.

On peut citer [Gang], à ce propos, pour l'étude faite en condition réelle sur les différentes structures pouvant être employés dans l'implémentation du balayage.

L'algorithme de balayage peut s'appliquer à beaucoup d'autres problèmes géométriques qui n'ont pas été abordés ici: calcul de l'union et de l'intersection de figures, remplissage de figures par des couleur, etc...

Un des intérêts du balayage du plan est la simplification des objets manipulés. Un dessin formé de segments quelconques peut être transformé en un ensemble de sommets et d'arêtes ne se coupant pas deux à deux. On obtient ainsi, à partir d'un dessin formé de seg-

ments, un représentation en terme de graphe planaire (voir [Mois]). On peut ainsi simplifier des traitements ultérieurs sur le dessin comme le fenêtrage.

6. REFERENCES

[Bent] BENTLEY J.L., OTTMANN T.,Algorithm for reporting and counting geometric intersections, IEEE Trans. Comp., Vol.C-28, 1979, pp.643-647.

[Brow] BROWN K.Q., Comments on "algorithm for reporting and counting geometric intersections", IEEE Trans. Comp. Vol.C-29, 1981, pp.147-148.

[Gang] GANGNET M., MICHELUCCI D., Un outil graphique interactif, Rapport de recherche n°84-12, Ecole des Mines de Saint-Etienne, Octobre 1984.

[Habi] HABIB M., HAMROUN M., JEGOU R., Linear equivalence for transitivity in graphs, Rapport de recherche n°83-10, Ecole des Mines de Saint-Etienne, Mai 1983.

[Luca] LUCAS M., La réalisation des logiciels graphiques interactifs, CEA-EDF-INRIA, Ecole d'été d'informatique, Eyrolles, juillet 1979.

[Mois] MOISSINAC J.C., Aide informatique à la réalisation de dessins animées, Thèse de docteur-ingénieur, Ecole des Mines de Saint-Etienne, 1984.

[Nier] NIERVEGELT J., PREPARATA F.P., Plane sweep algorithm for intersecting geometric figures, Communication of the ACM, Vol.25, n°10, October 1982, pp.739-747.

[Sha1] SHAMOS M.I., Computational geometry, Ph.D. Thesis, Dept. of comput. Sci., Yale University, New-Haven, Conn., 1978.

[Sha2] SHAMOS M.I., HOEY D., Geometric intersection problems, in Proc. of the 17th Annual IEEE Symposium on Foundations of Computing, 1976, pp.208-215.

[Tous] TOUSSAINT G.T., Pattern recognition and geometrical complexity, Proc. 5th International Conference on Pattern Recognition, IEEE, 1980, pp.1324-1347.

[Vais] VAISHNAVI V.K., WOOD D., Rectilinear line segment intersection, layered trees and dynamization, J. of Algorithms, 3, 1982, pp.160-176.

[Wirt] WIRTH N., Program development by stepwise refinement, C. of ACM, Vol.14, n°4, April 1971, pp.221-227.

ANNEXE 1

GENERALISATION DE

L'ALGORITHME DU DRAPEAU HOLLANDAIS

Nous exposons une généralisation triviale de l'algorithme du drapeau hollandais de DIJKSTRA ([Dijk] p.111-116), à un nombre C fixé de couleurs.

Soit une rangée de boites numérotées de 1 à N. On donne:

P1: chaque boite contient un caillou.

P2: chaque caillou a une couleur choisie parmi C, $C \leqslant N$.

Le but est de réarranger les cailloux dans l'ordre des couleurs de 1 à C avec seulement les deux opérations suivantes:

échanger(i,j) $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant N$
si i=j on ne fait rien;
si $i \neq j$ on échange les deux cailloux des boites i et j;

couleur(i) $1 \leqslant i \leqslant N$
donne la couleur du caillou de la boite i;

On distingue C+1 catégories de cailloux:

- C catégories E_i de cailloux de couleur i déjà triés.

- 1 catégorie X de cailloux non triés.

L'ordre des différentes catégories est le suivant:

E_1 , E_2 , ... , E_k , X , E_{k+1} , ... , E_C avec k=|C/

Pour repérer les frontières entre ces catégories, on dispose d'un tableau IND de C indices:

$1 \leqslant i < IND[1]$ → i est dans la première catégorie
$IND[1] \leqslant i < IND[2]$ → i est dans la deuxième catégorie
..
$IND[C] \leqslant i \leqslant N$ → i est dans la dernière catégorie

Dans l'état initial, tous les cailloux seront dans la zone X, le tableau IND sera donc initialisé par:

$IND[i]=1$ pour $1 \leqslant i \leqslant |C/2|$
$IND[i]=N$ pour $|C/2|+1 \leqslant i \leqslant N$

Cet ordre sur les zones et le tableau IND représente, en fait, l'invariant de l'algorithme.

Dans l'état final, la zone X sera vide donc on aura

$$IND[|C/2|] = IND[|C/2|+1]$$

qui sera le test d'arrêt.

L'état final se produit de façon obligatoire, soit en augmentant $IND[|C/2|]$, soit en décrementant $IND[|C/2|+1]$ de un à chaque tour de boucle.

L'algorithme est décrit dans la suite.

```
pour i ← 1 à |C/2| faire IND[i] ← 1;
pour i ← |C/2|+1 à N faire IND[i] ← N;

tant que IND[|C/2|] < IND[|C/2|+1] faire
 début
 coul ← couleur(IND[|C/2|]);
 si coul≤|C/2|
   alors début
       i ← |C/2|;
       répéter
          échanger(IND[i-1]+1,IND[i]);
          IND[i] ← IND[i]+1;
          i ← i-1;
       jusqu'à i+1=coul;
       fin
   sinon début
       i ← |C/2|+1;
       répéter
          échanger(IND[i-1]+1,IND[i]);
          IND[i] ← IND[i]-1;
          i ← i+1;
       jusqu'à i-1=coul;
       fin;
fin;
```

Algorithme .1

Pour prouver cet algorithme, il suffit de prouver que l'invariant décrit ci-dessus est conservé par la boucle principale.

La boucle externe est exécuté N fois et pour chaque tour il ya au pire |C/2| échanges. La complexité est donc en O(CN) pour le nombre d'echanges. Il y a N appels à la fonction couleurs.

Cet algorithme peut implémenter l'opération diviser en remplaçant les cailloux par des points de la P_structure et la couleur par le segment au-dessus duquel se trouve chaque point.

Si k est le nombre de segments rendus par la fonction points_extrem et P le nombre de point de la P_structure, alors la "couleur" d'un point se calcule en O(k). La complexité de l'opération diviser est donc en O(k|P|).

ANNEXE 2

LE PARADIGME DU RATISSAGE

De nombreuses méthodes peuvent aider à la découverte et à l'écriture d'algorithmes et de leurs preuves.

On peut raisonner par induction en deçomposant un problème en sous-problèmes de même type. Il s'agit alors du principe "Divide-and-Conquer".

On peut aussi établir une induction plus simple dite du ratissage.

Les éléments de la donnée à traiter sont classés suivant un ordre que nous appellerons ordre d'exploration et l'on établit l'équation de récurrence qui permet , lorsque les i premiers élements ont été explorés, de traiter le i+1ième

L'algorithme du tri par insertion, du drapeau hollandais de DIJKSTRA, le balayage du plan entre dans cette catégorie.

On peut aussi citer aussi un problème posé par BENTLEY:

> La donnée est un vecteur de N nombres réels; le résultat est la somme maximum trouvée pour tous les sous-vecteurs contigus. L'algorithme trivial est en $O(N^3)$.

Il est à noter cependant que BENTLEY mentionne avoir cherché avec l'aide de SHAMOS une borne inférieure en $\Omega(n\log n)$, après avoir trouvé un algorithme basé sur "Divide-and-Conquer" de la même complexité.

Exposé quelque temps plus tard à un séminaire, à Carnegie-Mellon, ce problème fut résolu en une minute par Jay KADANE, qui fournit une solution optimale en $O(N)$ en appliquant le principe de ratissage. Et pour-

tant SHAMOS est l'inventeur du balayage du plan!

DEUXIEME PARTIE

SUR LA MANIPULATION INTERACTIVE DE GRAPHES ET D'ENSEMBLES ORDONNES

INTRODUCTION

UN OUTIL DE MANIPULATION DE GRAPHES

1. HISTORIQUE ET MOTIVATIONS

Les outils traditionnels du théoricien des graphes sont le crayon et la feuille de papier. L'élaboration de la plupart des conjectures et des théorèmes dans ce domaine passent par une étape de va et vient entre des dessins de graphes et des calculs sur ceux-ci.

La taille et le nombre de graphe qu'il est possible de traiter à la main lorsqu'on manipule des objets éminemment combinatoires (tel par exemple un polynôme chromatique de graphe) est limitée. Or travailler sur les tous premiers exemples d'une conjecture peut être insuffisant pour se faire une intuition solide sur cette conjecture. Constatant en outre que de nombreux combinatoristes travaillent maintenant à l'aide de micro-ordinateurs, il semble tout naturel de fournir à ces théoriciens un outil informatique, qui leur permette de manipuler des objets plus grands et de profiter d'un environnement informatique.

Il s'agit donc de réaliser un **logiciel** qui permette les **manipulations et calculs sur des objets combinatoires**, tels que l'on peut le faire sur un cahier de brouillon traditionnel ou un bloc-notes et qui sera destiné aux chercheurs de ce domaine, chercheurs qui ne sont pas obligatoirement informaticiens.

Nous avons choisi comme domaine d'application **les graphes et les ensembles ordonnés**, car nous y avons une certaine expérience de recherche et qu'au sein de notre équipe ce sujet est toujours d'actualité.

L'idée d'un tel outil est déjà relativement ancienne. Nous avons trouvé une thèse décrivant un tel système publiée en 1969 [Wolf]. On peut citer aussi le "Graph Manipulation Package" de R.C.READ [Read] en 1975.

Plus récemment, une équipe de l'Université de Belgrade a développé un logiciel ambitieux comprenant un démonstrateur de théorèmes et une base de données bibliographique [Cvet].

Notre travail se situe dans le prolongement d'une réalisation plus récente de l'équipe du professeur M.CHEIN au Centre de Recherche en Informatique de Montpellier [Bord], [Guid]. Ce dernier logiciel, appellé **CABRI**, est un **CA**hier de **BR**ouillon **I**nformatisé.

Il est à remarquer que le travail, pionnier en France, de l'équipe de Montpellier et en particulier celui de Y.GUIDO nous a été d'une grande aide et a largement motivé notre réalisation. De même, le travail de Y.GUIDO a influencé d'autres projets similaires qui viennent aussi de démarrer en 1984, au Mans (BOUCHET, FOUQUET) et à Grenoble (LABORDE, Laboratoire LSD) sur Macintosh et Lisa.

En outre les besoins des chercheurs pour de tels logiciels se sont manifestement accrus, depuis quelques temps [Laza]. L'équipement des laboratoires en traitement de textes sur micro-ordinateur a habitué les scientifiques à être présents un bon nombre d'heures devant un terminal, et les réticences vis à vis de l'informatique se sont un peu estompées.

Ce logiciel sera appellé dans la suite **KABRI** pour le distinguer de CABRI, développé à Montpellier.

2. *PLAN DU RAPPORT*

Les points suivants seront plus particulièrement abordés:

- Chapitre 1: **Développement de logiciels interactifs**
- Chapitre 2: **Description de KABRI**
- Chapitre 3: **Conclusions et perspectives.**

En étudiant, indépendamment des fonctionnalités pouvant être offertes, le **modèle d'interaction** pour KABRI, nous avons suivi les conseils de spécialistes en réalisation de systèmes interactifs, créateurs de STAR de XEROX dont sont tirés tous les concepts existants dans les très populaires micro-ordinateurs LISA et surtout MACINTOSH de APPLE.

> *"We have learned from Star the importance of formulating the fundamental concepts (the user's conceptual model) **before** software is written, rather than tacking on a user interface **afterward.** Xerox devoted about thirty work-years to the design of Star user interface. It was designed **before** the functionality of the system was fully decided. It was designed **before** the computer hardware was built. We worked for two years **before** we wrote a single line of actual product software. Jonathan Seybold put it this way,"Most system design efforts start with hardware specifications, follow this with a set of functional specifications for the software, then try to figure out a logical user interface and command structure. The Star project started the other way around: the paramount concern was to define a conceptual model of how the user would relate to the system. Hardware and software followed from this"[Seyb]" (tiré de [Smit]).*

Le chapitre 1 décrit ce qui a donc été la première étape de notre réflexion.

Il contient une méthode de développement d'une interface de communication homme-machine. Cette méthode est caractérisée par trois étapes :

- Analyse des tâches dans l'environnement traditionnel (non informatique) de l'utilisateur.

- Description sémantique des tâches dans un environnement informatique. Il s'agit de la description du modèle d'utilisation du logiciel.

- Description syntaxique constituée par la représentation externe des données et des résultats et la syntaxe de chaque commande.

Les références des documents qui nous ont guidés dans cette réflexion sur ce modèle peuvent être trouvées dans la bibliographie commentée en annexe.

La réalisation d'un logiciel interactif exclut a priori une spécification complète et exacte de tout le logiciel.

On adopte donc en général **une démarche expérimentale** procédant par la réalisation de prototypes successifs et leurs validations auprès des utilisateurs.

Cette démarche expérimentale demande une implantation modulaire, garantissant l'indépendance entre les fonctionnalités, l'interface de communication et les

périphériques employés afin de faciliter les modifications nécessaires demandées par l'utilisateur.

L'interactivité implique aussi des possibilités d'extension pour qu'un utilisateur puisse adapter lui-même le logiciel à ses problèmes spécifiques.

Plusieurs solutions sont proposées et discutées pour réaliser un logiciel extensible : définition de macro-opérations, incorporation de programmes écrits par l'utilisateur soit grâce à un éditeur de liens dynamique, soit par communication entre processus,...

Dans le premier paragraphe du chapitre 2, nous reprenons une analyse du prototype CABRI et nous proposons un autre modèle d'interaction pour KABRI, basé sur l'analyse des tâches du chercheur en théorie des graphes.

. Le chapitre 2 contient donc l'analyse du travail d'un théoricien des graphes.

Les différents types de travaux sont:

- dessin de graphes géométriques
- calcul de propriétés et d'invariants
- écriture de notes, d'idées, de texte d'articles
- démonstration de nouvelles propriétés

Nos effort ont été portés essentiellement sur la partie "construction de graphes géométriques", par l'étude d'un éditeur de dessins de graphes.

Le domaine de la théorie des graphes est très large et en constante évolution. Pour assurer sa survie, KABRI devait être **extensible** et aisément **modifiable**, et ceci même par un utilisateur non spécialiste.

Deux choix ont été faits:

- l'implantation du cahier de brouillon sera faite dans un langage interprété (en l'occurence Lisp) plutôt que compilé.
- les fonctions de calcul spécifiques à un utilisateur, seront réalisées sous forme de "processus indépendants" de Kabri.

Le chapitre 3 contient un rappel des idées importantes soulevées par cette réalisation ainsi que des idées d'extensions possibles dans un stade de développement ultérieur.

Le logiciel KABRI n'est actuellement qu'un prototype de logiciel de manipulation de graphes et d'ensembles ordonnés.

Ainsi, la version actuelle, réalisée avec le concours de M. Dao et J.P Richard du CNET, a permis de tester une implantation rudimentaire en C [Kern] sous UNIX (nous ne disposons pas actuellement de la version graphique de Le_Lisp nécessaire). Cette version utilise un système de multifenêtrage expérimental, APOTRE. Ici commencent les problèmes. APOTRE n'est disponible que avec une version modifiée UNIX (qui s'appelle MPX) qui n'est pas celle employée au département informatique de l'Ecole des Mines. Le développement s'est donc déroulé en vacation avec les deux systèmes. APOTRE ne dispose pas de bibliothèques de programmes pour le graphique. Le graphique a donc été réalisé par une bibliothèque indépendante écrite en assembleur.

Cette version a été présentée aux troisièmes rencontres de la R.C.P. 698 (18-20 avril 1985) sur "Les ensembles ordonnés et leurs applications".

Une communication et une démonstration ont été proposées pour les "Journées SM90" organisées conjointement par le CNET (Centre National d'Etude des Télécommunication), l'INRIA (Institut National de Recherche en Informatique et en Automatique) et l'ADI (Agence De l'Informatique), le 4,5,6 décembre 1985 à Versailles.

Aujourd'hui, l'intérêt autour de CABRI a pris de l'ampleur. D'autres équipes de recherche se sont associées, entre autres le Laboratoire Structures Discrètes (UA 393) de l'IMAG (Institut de Mathématiques Appliquées de Grenoble). Un projet national est en cours de rédaction dans le cadre du PRC "Mathématiques et Informatique", regroupant éventuellement les équipes de Combinatoire d'Orsay (LRI-UA410) et de Paris (ER 193), du centre de recherche en Informatique du Mans (UA 1099) (Voir l'annexe 3).

CHAPITRE 1

DEVELOPPEMENT DE LOGICIELS INTERACTIFS

1. ETUDE DU CABRI DE MONTPELLIER

La première réalisation de CABRI (CAhier de BRouillon Informatisé) et son analyse et critique ont été le sujet de la thèse de troisième cycle de Y.GUIDO [Guid].

Les défauts de cette réalisation, analysés par Y.GUIDO, apparaissent dès les premières utilisations de ce logiciel.

Le dialogue est réalisé à travers un système de menus hiérarchisé selon la décomposition modulaire du programme:

- fonction d'édition de graphes
- fonction de calcul sur ces graphes
- fonction de gestion et de sauvegarde de ces graphes

Ce choix rend certaines manipulations, que l'on voudrait courantes, malaisées, par exemple dans le passage du module de calcul au module d'édition de graphes. Or il apparaît à l'usage que l'utilisateur veut modifier son graphe rapidement entre deux calculs.

Nous remarquons ainsi que la structuration d'une interface calquée sur la décomposition fonctionnelle du logiciel n'est pas obligatoirement très efficace, les buts poursuivis dans les deux cas n'étant certainement pas les mêmes.

Y.GUIDO propose pour résoudre ce problème une nouvelle répartition des commandes, c'est à dire d'un côté les commandes de gestion de la mémoire et de l'autre les commandes d'édition et de calcul en un seul niveau, ne suivant plus le découpage modulaire du logiciel.

Cette hiérachisation des menus interdit à l'utilisateur d'automatiser l'enchaînement de fonctions existantes et donc d'en fabriquer de nouvelles sans avoir à modfier CABRI lui-même.

Un logiciel tel que CABRI ne peut se contenter d'un nombre restreint de fonctions de calcul et d'édition, le domaine de l'algorithmique des graphes et des ordres étant en constante évolution.

De plus, chaque utilisateur doit pouvoir utiliser facilement des fonctions personnelles. La possibilité d'étendre CABRI est donc indispensable à son utilisation et à son développement. Celle-ci doit pouvoir être réalisée de manière simple et souple. On ne peut demander à chaque utilisateur d'être un spécialiste de l'informatique et de CABRI. On doit pouvoir accepter différentes formes de programmes (langages, structure de données,..) avec le minimum de modification. Y.GUIDO signale ce problème qu'il lie à un insuffisance logicielle du Pascal UCSD, mais il ne propose apparemment pas de solution.

Y.GUIDO souligne le manque général d'organisation et de puissance de la gestion des graphes et signale qu'elle devrait évoluer vers un genre de base de données, permettant ainsi à l'utilisateur une gestion des graphes par propriétés, qui serait en effet plus adaptée.

Cette thèse met l'accent sur les difficultés soulevées par une telle réalisation. Celles-ci sont dues en partie au choix de l'implémentation sur micro-ordinateur (THEMIS, Apple II,..). Ces machines n'offrent pas en général un environnement logiciel (ici Pascal UCSD) et matériel agréable pour accueillir un tel projet, réalisé par un non-spécialiste des systèmes d'exploitation informatiques.

Ainsi, les moyens de saisie et d'affichage disponibles pour CABRI sont trop inconfortables et peu souple (écran à mémoire type Tektronix, trop grand usage du clavier alphanumérique par manque de périphérique de désignation tel que souris,joystick,..).

La suite du chapitre contient la description d'une nouvelle approche destinée à diminuer ces inconvénients.

2. *INTERFACE HOMME-MACHINE*

La réussite du développement d'un logiciel dépend d'une bonne estimation de ses performances.

Les performances en temps et en mémoire sont en général bien estimées. Mais on oublie trop souvent de tenir compte des performances de l'interface de communication avec l'utilisateur.

Nous appelerons **interface utilisateur**, tous les aspects du logiciel en contact avec l'utilisateur, **physiquement** et **conceptuellement**. Les autres aspects du logiciel, qui sont cachés à l'utilisateur, sont ce que l'on peut considérer comme **l'implantation**.

Cette séparation théorique n'est pas en général aussi claire dans la pratique; souvent des aspects de l'implantation que le programmeur pense cachés, réapparaissent dans l'interface.

Les performances d'une interface peuvent être mesurées suivant plusieurs critères:

- rapidité d'exécution d'une tâche
- gravité des erreurs de manipulation
- étendue du domaine des tâches pouvant être réalisées
- durée d'apprentissage
- facilité de mémorisation des manipulations pour l'utilisateur
- accord avec les habitudes de travail de l'utilisateur

Cette liste n'est pas exhaustive. De plus, on doit tenir compte de l'expérience de l'utilisateur dans l'utilisation d'un système informatique.

Plusieurs auteurs, comme MORAN [Mora] ou GREEN [Gree], ont mis au point des méthodes de spécification formelle d'interfaces. Mais le comportement et la psychologie des utilisateurs semble résister à toutes formalisations. Il n'y a donc pas actuellement de méthode systématique pour créer la meilleure interface possible. Les concepteurs font en général confiance à leur intuition.

Plusieurs approches sont en général utilisées:

- On examine les logiciels déjà existants, afin d'obtenir une liste de leurs caractéristiques. On fabrique alors une interface à partir des meilleures de celles-ci.

- On spécifie les fonctions une par une, et on implante chacunes de celles-ci en définissant une commande pour l'invoquer.

Ces deux approches fournissent en général des interfaces inconsistantes, c'est à dire sans logique d'ensemble. Une autre approche peut être :

- On implante un ensemble complet de fonctions et on élabore un langage de commandes consistant.

Cette dernière approche peut donner naissance à des concepts et abstractions inaccessibles à l'utilisateur moyen.

Pourtant un logiciel, dont l'interface utilisateur est mal construite, peut perdre tout intérêt car son apprentissage est trop compliqué. Dans le pire des cas, il peut être inutilisable même par des gens expérimentés. Les utilisateurs en sont amenés à suivre des "recettes de cuisines" sans pouvoir comprendre le sens de leurs manipulations et ils ne pourront jamais utiliser toutes les possibilités du logiciel même si celles-ci sont immenses.

D'où l'importance d'essayer d'élaborer une nouvelle stratégie pour guider la réflexion sur le développement d'une interface et d'éviter les erreurs les plus graves.

Cette stratégie comporte trois étapes.

Toutes les approches, énumérées ci-dessus, omettent une étape essentielle qui est **l'analyse des tâches** et de **l'environnement de travail** habituels de l'utilisateur.

Cette analyse comporte les points suivants:

- quels utilisateurs?
- quelles sont les tâches à accomplir?
- de quelles informations a-t-on besoin?
- quelles informations sont générées?
- quelles méthodes sont employées?

De cette analyse, on détermine une structuration du domaine des tâches en sous-tâches.

Elle induit une description des objets manipulés par l'utilisateur.

Cette analyse est un processus difficile à mener à bien. Elle doit se faire en collaboration avec des experts du domaine visé. KABRI s'est développé au sein de l'équipe "Algorithmique et Programmation", qui com-

prenait durant cette période des chercheurs en théorie des graphes et des ensembles ordonnées, ainsi que des praticiens réalisant des applications algorithmiques.

L'étape suivante est la **description sémantique** de l'interface.

Ce niveau représente ce que beaucoup d'auteurs appelle **"mental user's model"**. Elle représente l'ensemble des entités et des opérations sur ces entités permettant à l'utilisateur d'accomplir les tâches répertoriées au niveau précédent, dans l'environnement informatique.

Il est important de s'assurer de la consistance de cette description avec l'analyse des tâches. Pour chaque tâche répertoriée dans le domaine, on doit pouvoir trouver une méthode de réalisation en termes d'opérations atomiques. Cette description devra tenir compte des impératifs de complexité d'exécution d'une tâche (plus la méthode de réalisation sera complexe, moins l'interface sera efficace).

C'est une étape importante. Ce modèle, expliqué à l'utilisateur, lui permettra d'anticiper le comportement du système. Il sera donc utile dans la rédaction des manuels d'utilisation.

On passe ensuite à la **description syntaxique** de l'interface.

Elle est constituée par la représentation externe des données, la descrition de l'affichage (écho, message d'erreur, ...) et par la description précise de chaque commande.

Elle représente l'aspect concret du modèle d'interface.

La représentation des données doit être aussi fidèle (c'est le principe WYGIWYS "What You Get Is What You See") et économique que possible. Toutes les actions effectuées doivent avoir un effet visible et cohérent afin que l'utilisateur puisse prévoir la prochaine commande. Cette représentation doit être fidèle économique et non ambiguë. On trouvera beaucoup d'idées sur la représentation des données dans [Ber1] [Ber2].

Un exemple de représentation ambiguë est celle employée dans la version de "MAKE", utilitaire UNIX de maintenance de logiciel,dont nous disposons à l'école des mines, pour lequel tabulation et blancs ont des significations différentes malgré qu'ils ne soient pas différents une fois affichés.

Ce n'est qu'alors que l'on définit le style de dialogue en reliant chaque commande à des actions physiques. Le choix de ce style est fortement influencé par le type d'utilisateur du logiciel (novice ou expert) et doit s'accorder à ses habitudes de travail.

Ce dialogue doit être consistant (les mêmes actions doivent toujours avoir les mêmes résultats), demander le minimum d'effort (les commandes les plus employées doivent être les plus simples), sur (les commandes dangereuses doivent soit demander une confirmation, soit être réversibles).

Si l'environnement le permet, cette description peut être une simulation des outils habituels (non informatiques) de l'utilisateur.

Les avantages de cette stratégie sont les suivants:

- Le concepteur du logiciel dirige son attention sur le système vu par l'utilisateur.
- Elle permet d'assurer un certaine consistance et simplicité. Elle sert d'outil pour faciliter l'implémentation, la rédaction de manuels d'utilisation.
- Pour une analyse des tâches, le concepteur a la possibilité de décrire plusieurs modèles pour l'utilisateur ou plusieurs syntaxes d'interaction. Elle autorise donc une analyse des performances de l'interface (temps d'apprentissage, rapidité d'execution,..) grâce aux travaux en ergonomie et en psychologie appliquée [Car3].

Grâce à l'approche descendante que cette stratégie préconise, celle-ci permet d'attirer l'attention du concepteur sur le domaine des tâches que doit supporter le logiciel et sur le modèle conceptuel, qu'aura l'utilisateur, de ce logiciel.

Trop porter l'attention sur le style d'interaction peut cacher l'importance du modèle sous-jacent du logiciel. L'élaboration de ce modèle nous semble le point le plus important de la stratégie proposée. Sans celui-ci, le logiciel peut devenir inutilisable soit par une trop grande complexité et un manque d'uniformité (utilisation style recette de cuisine), soit à cause de la difficulté d'apprentissage et de mémorisation.

Comme on peut le voir, une bonne interface ne peut être une verrue rajoutée à un programme déjà existant, mais doit être prise en compte dés le début de la

réflexion.

Cette stratégie reste informelle; elle n'offre ni langage de spécification, ni méthode de vérification de consistance et d'efficacité. Une telle étude sortirait du domaine de ce rapport.

La description de l'interface de KABRI suivra cette stratégie.

3. MODULARITE

Le développement d'un logiciel interactif est, pour une part, une activité expérimentale (dans l'état actuel des connaissances) comme nous l'avons signalé dans l'introduction.

On doit donc faciliter les modifications du logiciel durant son cycle de vie, grâce à une décomposition bien choisie séparant fonctionnalités et interface, afin de permettre une adaptation au type d'utilisateur. On peut ainsi préférer une interface basée sur un système de menus pour des utilisateurs occasionnels à une interface de type langage de commande préférée par des experts informaticiens.

L'adaptation à des environnements physiques différents suppose aussi une séparation entre interface et périphériques. De nouveaux périphériques, de plus en plus sophistiqués apparaissent de jour en jour (bitmap couleur, synthétiseur et analyseur de la parole,...). Ces périphériques sont souvent disponibles sur des micro-ordinateurs. La plupart du temps, les systèmes d'exploitation ne permettent d'utiliser uniquement des terminaux vidéos avec des claviers QWERTY. La gestion de terminaux possédant des protocoles plus sophistiqués est assurée par le logiciel d'application.

De manière traditionnelle, les logiciels sont décomposés en modules, qui chacun réalise une ou un ensemble de fonctions. Une telle approche tend à ignorer l'importance de l'interface de communication avec l'utilisateur. Si chacun des modules est réalisé par un programmeur différent, l'interface aura des chances d'être inconsistante. Si un module est modifié sans toucher aux autres, Il risque d'arriver la même mésaventure et que l'interface devienne incohérente.

Les protocoles nécessaires au contrôle des terminaux sophistiqués ne sont pas, en général, soutenus par les systèmes d'exploitation, mais par le logiciel d'application lui-même. Chaque type de terminaux ayant son propre protocole, le logiciel devient de plus en plus complexe s'il doit fonctionner dans plusieurs environnements. Cela restreint fortement les possibilités d'amélioration de l'interface grâce à un matériel

plus sophistiqué.

On propose une autre décomposition tenant compte de l'interface, différente de la décomposition fonctionnelle.

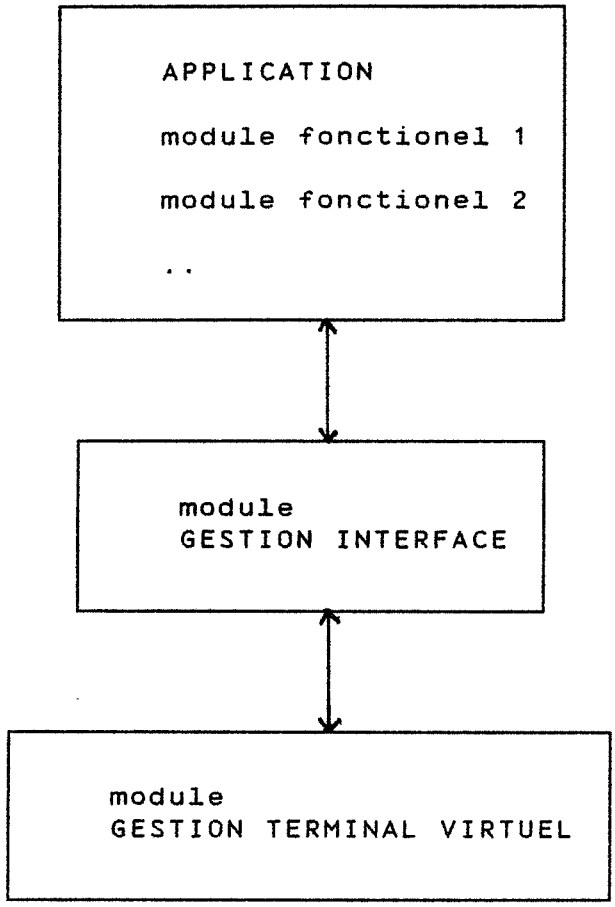
Un logiciel est découpé en modules, entre autre, dans le but de minimiser les effets d'une modification locale, sur le logiciel tout entier.

Les interfaces entre modules doivent être définies clairement et de façon générale pour ne pas avoir à être modifiées par la suite. On peut employer à cet effet la méthode de spécification par types abstraits [Gutt] que nous avons décrit dans la première partie de ce rapport. Un module est alors défini en termes d'objets et d'opérations sur ces objets.

Trois aspects d'un logiciel interactif sont amenés à être modifiés :

- l'ensemble des fonctionnalités, soit par la modification d'une fonctionnalité déjà existante, soit par l'ajout d'une nouvelle fonctionnalité.
- l'interface de communication, soit pour l'adapter à de nouvelles fonctionnalités, soit pour l'améliorer, soit pour l'adapter à un nouveau type d'utilisateur (novice ou expert).
- les périphériques utilisés.

La décomposition proposée est donc la suivante :



L'implantation de KABRI suivra cette décomposition modulaire.

4. EXTENSIBILITE

4.1. INTRODUCTION

Il est évident que l'extensibilité et la possibilité de modifications de l'environnement est une caractéristique des logiciels interactifs. L'utilisateur doit être le moins possible limité par les décisions des implémenteurs du logiciel.

Ce problème se pose de manière aiguë dans KABRI. En effet, le domaine de l'algorithmique des graphes et des ordres est trop large pour espérer inclure dans un seul logiciel les principales applications actuelles. En outre il est illusoire d'espérer maitriser toutes les applications possibles. Pour que KABRI puisse vivre et évoluer, il lui faudra l'aide des utilisateurs pouvant développer et insérer de nouvelles fonctionnalités.

Ainsi, KABRI est une structure logicielle interactive de manipulation de graphes et d'algorithmes de graphes.

Le problème d'extensibilité se pose essentiellement dans deux tâches :

- le dessin de graphes
- les calculs de propriétés sur les graphes

Dans la fabrication de dessins de graphes, le chercheur doit pouvoir utiliser des opérations plus complexes que l'ajout, la suppression ou le déplacement. **Par exemple, substituer à un noeud A un graphe G avec certaines règles de connexion entre les sommets voisins de A et les sommets de G.**

Le nombre sans cesse croissant d'algorithmes découverts (consulter par exemple le thèse de M. HAMROUN [Hamr]) dans le domaine des graphes et des ensembles ordonnés impose à KABRI la capacité d'accepter de nouvelles fonctions de calcul. L'idéal est de pouvoir aussi insérer des programmes déjà écrits avec le minimum de modifications.

Deux méthodes différentes, une pour l'édition de dessin de graphes et l'autre pour le calcul sur les graphes sont proposées.

4.2. EXTENSIBILITE DANS L'EDITION DE DESSIN

De la même façon que dans des logiciels du type EMACS ou VI, qui sont des éditeurs de texte, l'utilisateur peut définir des macro-opérations à partir des fonctions d'identification d'ajout et de suppression de sommets ou d'arcs.

4.3. EXTENSIBILITE DANS LES FONCTIONS DE CALCUL

L'utilisateur doit pouvoir disposer d'un ensemble de "bibliothèques" de fonctions de calcul. Chacune d'entre elles contiendra un nombre limité de fonctions différentes. Elles pourront être chargées dynamiquement quand le besoin s'en fait sentir. Ces bibliothèques devront pouvoir être fabriquées facilement à partir de programmes de provenances diverses.

Lors du chargement, il y a redéfinition des noms de fonctions déjà connues. On permet ainsi à l'utilisateur de redéfinir à sa guise certaines fonctions, et d'en employer de plus performantes.

Un système écrit en PASCAL, ou un autre langage compilé, ne peut pas avoir cette possibilité. Il devra être modifié et recompilé et une extension deviendra alors une version séparée. L'utilisateur devra alors choisir, avant de travailler la version qui lui convient, avec l'impossibilité de combiner des fonctionnalités de deux versions différentes. De plus, l'extensibilité n'est plus possible dynamiquement pendant une

session de travail.

Une solution est l'écriture du système grâce à un langage interprété du type LISP, APL ou SNOBOL. Données et fonctions ont alors la même forme (la forme des S-expressions pour LISP, par exemple), et les données peuvent être exécutées comme des programmes. Les fonctions, à charger dynamiquement doivent alors être écrites dans ce même langage ou, sinon, être intégrées dans dans le corps même de l'interprète et l'on revient à l'impossibilité précédente.

Il est aussi possible d'utiliser un éditeur de liens dynamique.

Mais un éditeur de liens dynamique est un programme compliqué à construire, lourd d'emploi et il y en a peu de disponibles (le seul que nous connaissons est celui disponible sur le système MULTICS). Aucun éditeur de ce type n'est disponible actuellement dans l'environnement logiciel UNIX dont nous disposons.

La solution que nous avons finalement retenue, est celle de l'exécution des fonctions par des processus indépendants créés dynamiquement au cours de l'exécution. Les données et résultats sont transmis par communication entre ces processus. Des facilités de ce type sont offertes par UNIX (voir annexe 3).

Les fonctions de calcul seront des programmes écrits dans un langage quelconque, ceci indépendamment de KABRI.

Un ensemble de procédures d'accès aux objets manipulés et à leur affichage sera disponible pour le créateur de ces nouvelles fonctions, cela pour chaque langage employé pour écrire ces fonctions. Ainsi l'utilisateur pourra ajouter des fonctionnalités à KABRI sans avoir à connaître sa structure interne.

Le type abstrait suivant décrit le genre de procédures pour accéder à un graphe de KABRI.

```
type graphe

paramètre : noeud.

constante : graphe_vide : graphe(noeud).

opérations :
   aj_noeud(graphe(noeud),noeud) → graphe(noeud)
   aj_arc(graphe(noeud),noeud,noeud) → graphe(noeud)
   enl_noeud(graphe(noeud),noeud) → graphe(noeud)
   enl_arc(graphe(noeud),noeud,noeud) → graphe(noeud)
   existe_noeud(graphe(noeud),noeud) → booleen
   existe_arc(graphe(noeud),noeud,noeud) → booleen
   vide(graphe(noeud)) → booleen
   nbre_noeud(graphe(noeud)) → entier
   nbre_arcs(graphe(noeud)) → entier

significations :
   enl_noeud(graphe_vide,N) → graphe_vide
   enl_noeud(aj_noeud(G,N1),N2) →
       si N1 = N2
           alors G
           sinon aj_noeud(enl_noeud(G,N2),N1)
   enl_arc(graphe_vide,N1,N2) → graphe_vide
   enl_arc(aj_arc(G,N1,N2),N3,N4) →
       si N1 = N3 et N2 = N4
           alors G
           sinon aj_arc(enl_arc(G,N3,N4),N1,N2)
   existe_noeud(graphe_vide,N) → faux
   existe_noeud(aj_noeud(G,N1),N2) →
       si N1 = N2
           alors G
           sinon aj_noeud(existe(G,N2),N1)
   existe_arc(graphe_vide,N1,N2) → faux
   existe_arc(aj_arc(G,N1,N2),N3,N4) →
       si N1 = N2 et N1 = N4
           alors G
           sinon aj_arc(existe_arc(G,N3,N4),N1,N2)
   nbre_noeud(graphe_vide) → 0
   nbre_noeud(aj_noeud(G,N)) → nbre_noeud(G) + 1
   nbre_arc(graphe_vide) → 0
   nbre_arc(aj_arc(G,N1,N2)) → nbre_arc(G) + 1
```

Lors de la reconnaissance d'une fonction par KABRI, il y a **création d'un processus fils**, par la primitive du système FORK, chargé de l'exécution de cette fonction, grâce àla primitive EXEC, la transmission des données et résultats se faisant par des tubes de communication (voir annexe 3).

Les avantages de cette méthode tiennent à la **facilité d'ajout d'une fonctionnalité** à KABRI à partir d'un programme quelconque déjà existant.

Passons aux désavantages :

- Le manque de rapidité dû à l'utilisation des tubes qui obligent à deux transformations:

structure dynamique	<=>	structure statique

dans l'état actuel de UNIX qui ne tolère pas que plusieurs processus partagent des variables.

- L'utilisation répétée de FORK et EXEC ralentira l'interprète. En un premier temps, nous avons décidé d'utiliser la primitive FORK une seule fois à l'initialisation de KABRI. Chaque fonction est par la suite, lancé grâce à la primitive EXEC par la fonction précedente. Ceci est réalisé en imposant à toute fonction de comporter comme dernière instruction, un appel à une procédure FIN. Cette procédure restera bloquée jusqu'à la demande par KABRI d'une exécution, puis écrasera son code par celui de la nouvelle fonction grâce à EXEC.

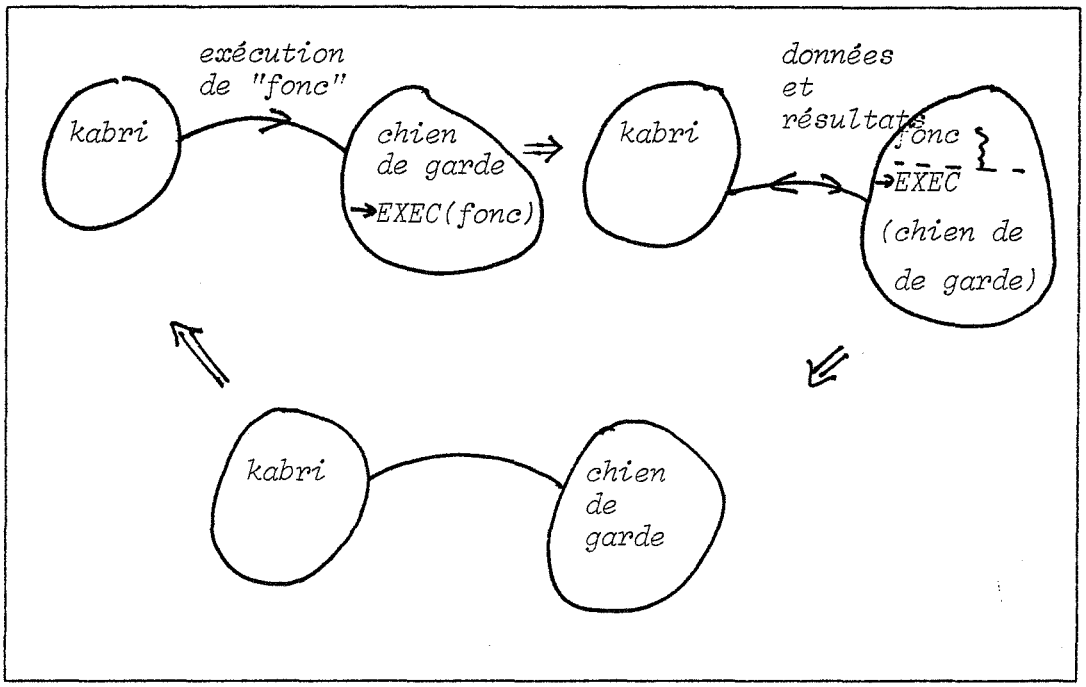

exécution
de "fonc"
kabri
chien
de garde
→EXEC(fonc)
kabri
données
et
résultats
fonc
→EXEC
(chien de
de garde)
kabri
chien
de
garde

Figure 4.3.1

CHAPITRE 2

DESCRIPTION DE KABRI

1. ANALYSE DES TACHES ET DES BESOINS

Le but poursuivi est d'analyser le travail d'un chercheur en théorie des graphes.

1.1. DEFINITIONS DES PRINCIPAUX TERMES

Nous définissons les principaux termes utilisés dans la suite de cette section.

graphe:

Un graphe G=(X,U) est le couple constitué:

a) par un ensemble de sommets X={x1,x2,...,xn}
b) par une famille d'arcs U=(u1,u2,...um) d'éléments du produit cartésien XxX={(x,y)/xϵX,yϵY};

dessin de graphe :

Un dessin de graphe est un schéma plan constitué par un ensemble (fini) de points, qui seront les sommets, et par ensemble de courbes simples du plan, orientées à l'aide d'une flèche, qui seront les arcs, reliant chacune deux de ceux-ci.

La forme des arcs et des sommets dessinés peut varier suivant les auteurs. Un nom et un nombre (poids) peuvent être associés à chacun des sommets et arcs. Des commentaires (texte) sont souvent associé à ce graphe.

La position des sommets est choisie, soit par une esthétique, soit par une propriété que l'on veut visualiser (les sommets d'un circuit seront

dessinés sur un cercle, par exemple). Ces dessins peuvent donc aussi être le résultat d'un calcul (par exemple un programme d'affichage d'un graphe planaire).

Pour certaines propriétés "non orientées" d'un graphe, la direction des flèches importe peu. Il n'y a alors que deux sommets reliés par une courbe, qui représentera une arête.

notes de travail :

Ces notes sont formées par un ensemble de textes non structuré. Elles sont liées aux dessins de graphes et aux calculs effectués.

Le notes et dessins sont "archivés" par le chercheur pour pouvoir être réutilisés dans la suite pour l'écriture d'un texte final.

texte scientifique :

Il s'agit d'un texte imprimé, mise en page suivant certaines règles (celles de la revue où doit être publié le texte, par exemple).

Ce texte comporte généralement des dessins de graphes et des formules mathématiques.

1.2. TACHE PRINCIPALE

Buts:

- Elaborations de conjectures.
- Démonstrations de théorèmes.
- production de textes scientifiques (rapports, articles,...).

Données:

Les données de ce travail sont informelles. Il s'agit d'idées, d'intuitions, de problèmes posés par d'autres chercheurs..... Le chercheur travaille sur des dessins de graphes, calcule les propriétés de ces graphes, sur ses notes de travail, sur des articles et des livres.

Resultats:

- Conjectures.
- Théorèmes démontrés.
- Notes de travail, communicables à des collègues.
- Rapports de recherche, articles comprenant à la fois dessins de graphes et formules scienti-

fiques.

Méthodes:

a) dessins de graphes sur une feuille de papier.
b) calculs sur ces graphes dessinés.
c) écriture de notes de travail sur un calepin.
d) archivage de ces dessins et de ces notes.
e) élaboration des théorèmes et de leurs démonstrations formelles à partir des trois étapes a, b, c et d.
f) écriture du texte à partir de e.

▸**Remarque** Les étapes a, b, c et d se déroulent en général dans un ordre quelconque et de manière discontinue dans le temps..

On obtient ainsi une division en six sous-tâches que nous allons décrire dans les paragraphes suivants.

1.3. MANIPULATION DE DESSINS DE GRAPHES

Buts:

Le chercheur construit des dessins de graphes sous un forme lui permettant de visualiser les propriétés de ce graphe.

Données:

- des feuilles, des crayons, une gomme
- un ou plusieurs graphes déjà dessinés ou non dont on peut connaître certaines propriétés déjà calculées.

Résultats:

Un ou plusieurs dessins de graphes dont certaines caractéristiques seront visibles et qui seront utilisés pour les calculs de propriétés ou d'invariants et pour l'écriture d'un texte. Ils sont donc en général archivés.

Méthodes:

a) Ajout, suppression et déplacement de sommets, d'arcs ou d'arêtes ou des opérations plus complexes (fusion, joint, ...) pouvant s'exprimer sous forme d'ajout et de suppression.
b) Mise en évidence visuelle (commentaires, formes, couleurs,..) de certaines parties (classes de sommets,...) de ces graphes.

1.4. CALCUL DE PROPRIETES ET D'INVARIANTS SUR CES GRAPHES

Buts:

Déterminer une ou les propriétés d'un ou plusieurs graphes.

Données:

- Un ou plusieurs graphes la plupart du temps sous formes de dessins.
- Des paramètres utiles au calcul (par exemple deux sommets A et B pour calculer le plus court chemin entre A et B).

Résultats:

- Un ou plusieurs graphes. Les objets de type circuits, chemins, composantes, peuvent s'interpréter comme des graphes. Ces graphes peuvent être sous la forme de dessins.
- Un invariant sous forme d'entiers ou de réels.
- Un propriété sous forme de booléen (vraie ou fausse).

Méthodes:

On applique les algorithmes connus ou venant d'être inventés "à la main". Dans le cas d'algorithmes exhaustifs, on essaye de se persuader d'avoir considéré tous les cas possibles.

1.5. ECRITURE DE NOTES DE TRAVAIL.

Buts:

Ecrire des notes consultables par soi-même ou communicables à un autre chercheur.

Garder la trace d'un certain travail.

Données:

- Des graphes, des dessins, des proprietés de ces graphes.
- D'autres notes déjà archivées.

Résultats:

Des notes manuscrites.

Méthodes:

En général, le chercheur utilise des feuilles de papier, liées parfois par des trombones

1.6. ARCHIVAGE DES DESSINS, DES NOTES ET DES PROPRIETES

Buts:

Notes, dessins de graphes et propriétés sont reliés et archivés afin pouvoir les consulter dans la suite du travail et pour la rédaction de rapports.

Données:

- notes de travail
- dessins de graphes
- resultats de calcul sur des graphes
- un critère de sélection

Résultats:

- notes et dessins de graphe correspondant au critère

Méthodes:

Cette recherche se fait en lisant le contenu des papiers où se trouve dessins et notes.

1.7. FABRICATION DE TEXTES DE RAPPORT

Buts:

Créer une version propre et définitive d'un texte résultant d'un travail.

Données:

- Dessins de graphes.
- Propriétés et résultats de calculs.
- Notes de travail.

Résultats:

Un ensemble de textes mis en page et imprimés comportant formules mathématiques et dessins de graphes.

Méthodes:

a) Fabrication d'un texte manuscrit.
b) Fabrication de dessins.
c) Mise en page des résultats de a et b grâce à une machine à écrire, des ciseaux, de la colle.
d) Si nécessaire, correction et retour à l'étape a.

2. QUELQUES REMARQUES

On peut noter la forte connexion entre les résultats et les données de ces différentes tâches. En effet, les tâches de dessins, de calcul et de prise de notes se déroulent en parallèle dans le travail du chercheur.

L'écriture du texte est plutôt une tâche effectuée en un seul temps, mais elle utilise tout le matériel amassé précédement.

L'utilisation d'un système de multifenêtrage permettra de conserver ce parallèlisme. Chaque tâche est associé à une fenêtre sur l'écran du poste de travail. Passer d'une tâche à une autre se résume à la designation de la bonne fenêtre. Ainsi d'un seul coup d'oeil, l'utilisateur considère le fonctionnement de plusieurs tâches qui vont lui permettre une prise de décision. Par exemple, plusieurs fenêtres, qui peuvent éventuellement se recouvrir, permettront l'édition de plusieurs graphes en même temps, chaque fenêtre étant considérée comme une feuille de papier sur un bureau.

La description du modèle de KABRI fera donc intervenir la notion de fenêtre.

Pour obtenir la même souplesse qu'avec un cahier de brouillon traditionnel, où aucune opération de modification n'est catastrophique (sauf la destruction d'une feuille de ce cahier), l'utilisateur devra, soit être prévenu du danger de la manipulation qu'il va effectuer, soit pouvoir défaire la dernière modification.

3. DESCRIPTION SEMANTIQUE

3.1. MODELE POUR L'UTILISATEUR

On décrit ici le modèle conceptuel de KABRI. Il s'agit d'abord de recenser les objets et opérations qui seront utilisés pour la réalisation des tâches déterminées par l'analyse, dans l'environnement informatique.

Ce modèle décrit pour l'instant seulement les tâches d'édition de dessins de graphes.

3.1.1. Description des objets manipulés par KABRI

Deux types d'objets seront manipulés par l'utilisateur:

- Les objets intinsèques à la tâche (par exemple les dessins de graphes).
- Les objets de contrôle servant à la manipulation du logiciel (les menus, les fenêtres, ...).

3.1.1.1. Objets intrinsèques

Dessins de graphes :

Un dessin de graphe est un schéma plan constitué par un ensemble de symboles (disques,carré,..), qui seront les sommets, et par un ensemble de courbes, qui seront les arcs, reliant chacune deux de ceux-ci. Il est dessiné par l'utilisateur dans une fenêtre d'édition (voir ci-dessous).

On appelle graphe actif, le graphe associé à la fenêtre active. **Il y en a au plus un sur lequel portera les commandes décrites ci-dessous.**

Chaque dessin est associé à un nom donné par l'utilisateur.

La forme des arcs et des sommets dessinés sera fixée par l'utilisateur. Un nom et un nombre (poids) peuvent être associés à chacun des sommets et arcs. Des commentaires sont souvent associés à ce graphe.

Noeud identifié :

Un noeud identifié passe en inverse vidéo. Il sera l'argument de la prochaine commande.

Arc ou arête identifié :

Idem que ci-dessus.

Classe de sommets :

Il s'agit d'un ensemble de sommets identifié. Ils seront les arguments de la prochaine commande.

Classe d'arcs ou d'arêtes :

Idem que ci-dessus.

3.1.1.2. Objets de contrôle

Fenêtres :

Une surface rectangulaire de l'écran. Les côtés de ces fenêtres sont parallèles au bord de l'écran. Elles sont utilisées pour afficher texte et graphique. Les fenêtres peuvent se superposer comme des feuilles de papier sur un bureau. Un ordre total est donc défini sur ces fenêtres.

Fenêtres d'édition de dessins de graphes :

Ces fenêtres sont utilisées pour dessiner des graphes. On peut donc leur appliquer toutes les opérations de construction de graphes. On peut en créer un nombre quelconque, mais une seule est active à la fois.

Il n'y a qu'un graphe associé à chaque fenêtre.

Fenêtre active :

La fenêtre sur laquelle peuvent s'appliquer les commandes explicitées ci-dessous. Il n'y a qu'une fenêtre active à chaque instant. Celle-ci est située au-dessus de toute les autres.

Surface sélectionnée :

Une portion de la surface de la fenêtre d'édition de dessin de graphe active. Elle est utilisée pour déterminer l'endroit et la taille d'un graphe à afficher.

Menus :

Un menu est une fenêtre constitué d'un certain nombre de lignes de texte, chacune correspondante à une commande autorisée.

Curseur :

Un curseur indique à chaque instant un point de l'écran sélectionné par l'utilisateur. Sa forme est caractéristique de la fenêtre à laquelle appartient ce point ou de l'opération en cours.

3.1.2. Description des opérations

On décrit ici toutes les opérations portant sur les objets définis ci-dessus.

3.1.2.1. Opérations sur les objets intrinsèques

Sélection d'un sommet, d'un arc ou d'une arête :

On sélectionne un de ces trois objets en positionant le curseur à proximité. L'objet sélectionné passe en inverse vidéo. Cette sélection permet d'indiquer l'argument de la prochaine commande.

Cette sélection n'est valable que sur le graphe actif.

Sélection d'une classe de sommets, d'arcs ou d'arêtes :

On sélectionne plusieurs objets d'un même type (sommets, arcs ou arêtes). L'ensemble des objets sélectionné passe en inverse vidéo. Ils seront l'argument de la prochaine commande. Cette classe est perdue après l'exécution de la commande.

Cette sélection n'est valable que sur le graphe actif.

Sélection d'un dessin de graphe :

On peut sélectionner un dessin de graphe par son nom. Cette sélection permet d'indiquer l'argument de la prochaine commande.

Ajout d'un arc ou d'une arête :

On spécifie l'origine, puis l'extrémité de l'arc. Cette origine et cette extrémité peut être :

- soit un sommet sélectionné.
- soit un point quelconque à l'intérieur de la fenêtre d'édition. Un nouveau sommet est alors créé.
- soit un arc sélectionné. Dans ce cas, l'arc est coupé en deux par le nouveau sommet crée.

Ajout d'un sommet :

L'ajout d'un sommet est équivalent à la création d'un arc dont l'origine et l'extrémité sont confondues. On ne peut pas créer un sommet sur un point de l'écran déjà occupé par un sommet.

L'ajout d'un sommet sur un arc coupe celui-ci en deux.

Suppression d'un sommet, d'un arc, ou d'une arête :

On peut supprimer un sommet, un arc ou une arête sélectionné. La suppression d'un sommet entraine la suppression de tous les arcs ou arêtes adjacents.

Déplacement d'un sommet :

Le sommet sélectionné est déplacé à la nouvelle position indiquée par l'utilisateur.

Si cette nouvelle position est occupée par un sommet, il y a fusion des deux sommets. Si elle est occupée par un arc ou une arête, celui-ci ou celle-ci est coupée en deux par le sommet.

Ajout d'un nom , d'un poids ou d'un commentaire :

Un nom, un poids ou un commentaire peuvent être ajouté à un sommet, un arc ou une classe de sommets ou d'arcs qui a été précédemment sélectionnée.

Opérations sur une classe :

On peut supprimer ou déplacer une classe de sommets. On peut supprimer une classe d'arcs ou d'arêtes.

Opérations sur un graphe :

Un nom peut être associé à un dessin de graphe.

On peut ajouter un dessin de graphe déjà construit au graphe associé à la fenêtre active. La position de ce dessin est donnée par la sélection d'une surface rectangulaire sur la fenêtre et d'un dessin par son nom.

On peut déplacer un dessin entier dans les limites de la fenêtre.

Opération défaire :

Cette opération permet de revenir sur la dernière modification effectuée sur le graphe. L'opération "défaire" peut elle-même être défaite.

3.1.2.2. Opérations sur les objets de contrôle

Création d'une fenêtre d'édition de graphe :

Une nouvelle fenêtre est créée de la dimension et à l'emplacement indiqué par l'utilisateur. Elle est associée à un dessin de graphe déjà existant ou vide. Cette fenêtre devient la fenêtre active et elle recouvre toutes les autres.

Suppression d'une fenêtre d'édition de graphe :

La fenêtre disparaît de l'écran et son contenu est perdu. Les fenêtres cachées réapparaissent.

Déplacement d'une fenêtre d'édition de graphe :

La fenêtre indiquée est déplacée à un nouvelle position de l'écran. Elle devient la fenêtre active et recouvre toutes les autres.

Activation d'une fenêtre d'édition de graphe :

La fenêtre désignée devient la fenêtre active et elle est placée au-dessus de toutes les autres.

Selection d'une surface sur une fenêtre d'édition :

Une surface peut être sélectionnée à l'intérieur d'une fenêtre d'édition de dessin, en désignant deux coins opposés.

Sélection dans un menu

Chacune de des lignes du menu correspond à une commande pouvant être désignée. Avant la sélection, la commande désignée est affichée en inverse vidéo. Après la sélection, le menu disparaît de l'écran.

4. DESCRIPTION SYNTAXIQUE

4.1. REPRESENTATION DES DONNEES

Ce paragraphe décrit comment les objets manipulés sont présentés à l'utilisateur.

4.1.1. Objets intrinsèques

Un dessin de graphe est représenté comme une image noir et blanc. Les noeuds sont représentés sous différentes formes (cercle, carré rectangle) de plusieurs tailles. Les arcs sont représentés sous la forme de segments avec une flèche indiquant la direction. Les arêtes sont representées par des segments. Ces segments peuvent avoir deux styles : continu et pointillé.

Un noeud identifié est rempli en blanc. Un arc ou une arête identifié est tracé en double.

4.1.2. Objets de contrôle

Une fenêtre est montrée comme un rectangle. Elle possède une zone d'affichage et une zone titre.

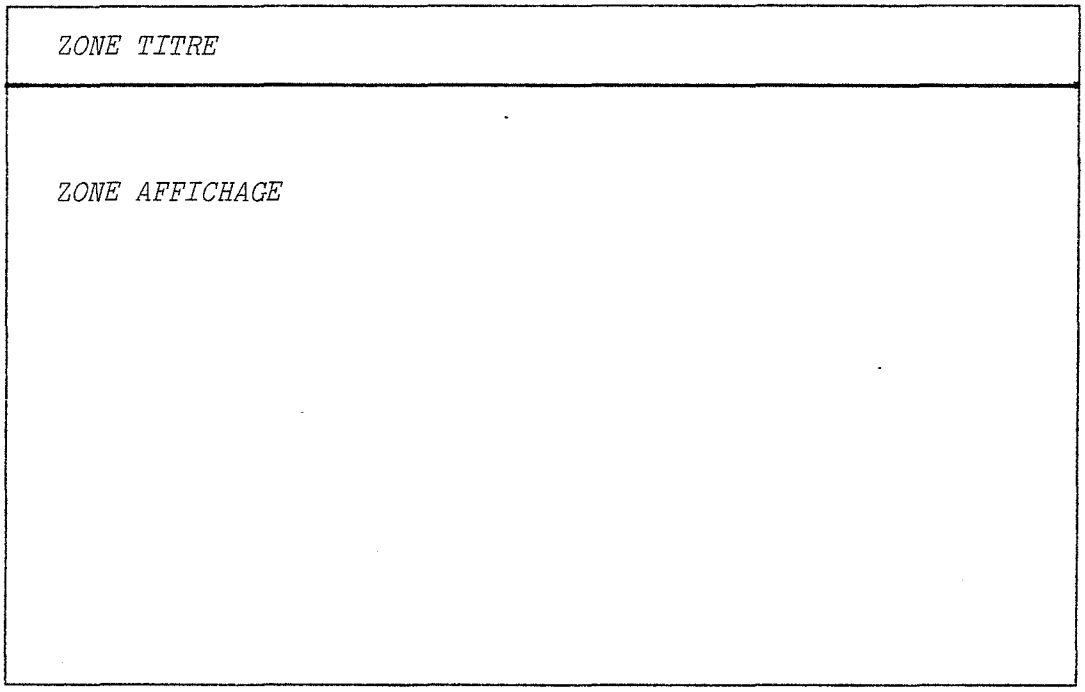


Figure 4.1.2.1

Une fenêtre d'édition de dessin a pour titre "édition du graphe" suivi du nom du dessin du graphe. La fenêtre de calcul a pour titre "calcul".

La fenêtre active est distinguée des autres par sa zone titre en inverse vidéo.

Quand une fenêtre est déplacée, seule la bordure suit le mouvement du curseur. Quand le déplacement est fini, la fenêtre est effacée et réaffichée au nouvel emplacement.

Une surface sélectionnée est montrée comme un rectangle aux bords parallèles aux côtés de la fenêtre.

Un menu est une fenêtre constituée d'une zone titre désignant le type d'opérations et d'un ensemble de lignes, une pour chaque commande. La commande désignée passe en inverse vidéo.

Un curseur est affiché continuellement. Il suit les mouvements de la souris. Son aspect change suivant la fenêtre dans laquelle il se trouve.

Lors de la création d'une arête, le curseur prend la forme d'un ligne elastique dont l'origine est fixée au début de l'arc et l'extrémité à la position de la souris.

Lors du déplacement d'un sommet, le curseur prend la forme du noeud déplacé et de l'ensemble des arcs adjacents. Ces arcs deviennent élastiques et suivent les déplacement de la souris.

Lors de la sélection d'une surface, le curseur prend la forme d'un rectangle élatique dont l'un des coins est fixe et le coin opposé suit les mouvements de la souris.

4.2. DESCRIPTION DES PERIPHERIQUES ET DE LEURS UTILISATIONS

Nous décrivons dans ce paragraphe l'ensemble des périphériques du poste de travail utilisé pour la communication entre l'utilisateur et le logiciel.

L'affichage se fait sur un bitmap haute résolution (780x1024 point)

Un bitmap est composé de trois éléments:

- une mémoire, appellée mémoire d'image, qui contient l'image affichée sous forme d'une matrice d'intensité
- un moniteur vidéo
- un controleur, qui transforme, environ 50 fois par seconde, le contenu de la mémoire d'image en un signal vidéo accepté par le moniteur

Un clavier traditionel (type QWERTY) permet à l'utilisateur déntrer ces textes et commandes.

Le système de multifenêtrage permet de simuler un ensemble de terminaux graphiques sur le bitmap, chaque fenêtre peut être considérée comme un terminal. Les entrées au clavier sont adressées à la fenêtre dans laquelle se trouve le curseur. S'il n'y a qu'une fenêtre présente sur l'écran, les entrées sont dirigées obligatoirement sur cette fenêtre.

Le poste de travail comporte aussi une souris munie de trois boutons. Cette souris sert à la désignation et au tracé.

La souris est une petite boite reposant sur une bille. Elle a été mise au point au Stanford Research Institute. Quand la souris est déplacée sur une surface plane, son mouvement dans les deux directions orthogonales est transmis à l'ordinateur par la rotation de la bille. Trois boutons, qui seront désignés par les nom droite, milieu et gauche, sont montés sur le dessus de la souris et l'utilisateur peut les actionner tout en déplaçant la souris. La souris est un outil de désignation très efficace comme le prouve diverses expérimentations [Car1].

Le déplacement du curseur sur l'écran est asservi à celui de la souris.

La souris permet de réaliser les opérations suivantes:

- Positionner le curseur sur un point de l'écran.

- Sélectionner un sommet, un arc ou une arête d'un dessin de graphe en positionnant le curseur à proximité.

- Sélectionner une commande dans un menu en positionnant le curseur sur la commande en en appuyant sur n'importe quel bouton.

- Tracer un arc en positionnant la souris, en appuyant sur le bouton gauche, en déplaçant la souris, puis en relachant le bouton.

- désigner un surface rectangulaire. il désigne d'abord un coin du rectangle, puis en maintenant un des trois boutons enfoncé, il déplace la souris. Un rectangle élastique se dessine, le coin diagonalement opposé suivant les déplacement de la souris. Le rectangle est fixé quand on relâche le bouton.

4.3. DESCRIPTION DES COMMANDES

La syntaxe de chaque commande sera décrite sous la forme :

nom de la commande

description informelle

1) entrée numéro 1 et état 1 → effet
2) entrée numéro 2 et état 2 → effet
...

Nous en décrirons qu'une partie pour ne pas alourdir le texte.

Ajout d'un sommet :

On déplace le curseur jusqu'à l'endroit désiré et on appuie et relâche le bouton gauche de la souris. Un noeud est alors crée. Si le noeud est positionné sur un arc, celui-ci est coupé en deux.

```
créer un sommet en (x,y)

1) déplacer la souris jusqu'en (x,y) et appuyer
et relâcher le bouton gauche
 → si le point (x,y) appartient à la fenêtre
         d'édition active
      alors
       si on identifie un noeud en (x,y)
         alors fini
         sinon
         si on identifie un arc AB passant par
            (x,y)
           alors
           début
           on supprime l'arc AB
           dans le graphe actif
           et on efface AB
           on crée un sommet C
           dans le graphe actif
           et on affiche C
           on crée l'arc AC dans le graphe actif
           et on affiche AC
           on crée l'arc CB dans le graphe actif
           et on affiche AC
           fin
           sinon on crée le sommet A
           dans le graphe actif
                 et on affiche A.
      sinon fini
```

Ajout d'un arc :

On déplace le curseur jusqu'à l'endroit désiré pour l'origine de l'arc. On appuie sur le bouton gauche puis en maitenant le bouton enfoncé, on déplace la souris la souris jusqu'à l'endroit desiré pour l'extrémité. Pendant le déplacement, un ligne élastique est créé entre l'origine de l'arc et la position de la souris.

```
créer un arc entre (x1,y1) et (x2,y2)

1) déplacer la souris jusqu'en (x1,y1) et
appuyer sur le bouton gauche.
 → si (x1,y1) appartient à la fenêtre d'édition
```

```
active
        alors on crée un sommet A en (x1,y1).

2) répéter , déplacer la souris en (xi, yi) dans
la fenêtre active
 → effacer le ligne élastique et la réafficher
entre (x1,x2) et (xi,yi)

3) relâcher le bouton en (x2,y2)
 → si (x2,y2) appartient à la fenêtre d'édition
active
      alors
      début
      on crée un sommet B en (x2,y2).
      on crée l'arc AB dans le graphe actif
      et on affiche AB
      fin
```

Suppression d'un sommet :

```
On déplace la souris jusqu'à proximité du sommet
et on appuie et on relâche le bouton du milieu.
le sommet et les arcs adjacents à ce sommet sont
alors supprimés.

Supprimer le sommet en (x,y)

1) déplacer la souris jusqu'en (x,y) et appuyer
et relâcher le bouton du milieu
 →si (x,y) appartient à la fenêtre d'édition
active
        alors
        si on identifie un sommet A en (x,y)
          alors
          début
          supprimer le sommet A
          dans le graphe actif
          supprimer tous les arcs adjacent à A
          effacer le graphe actif
          afficher le graphe actif
          fin
          sinon fini
        sinon fini
```

Toutes les autres commandes décrites dans le paragraphe précédent peuvent être aussi formalisée de cette façon.

5. ENVIRONNEMENT LOGICIEL D'IMPLANTATION

Le prototype de KABRI est réalisé sur une SM90, muni d'un micro-processeur 68000, et ayant comme système d'exploitation MPX, developpé au CNET, qui est un "UNIX-like".

Cette machine est munie d'un bitmap haute-résolution (780x1024 pixel) noir et blanc NUMELEC EHR90 et d'une souris possèdant trois touches de fonction.

KABRI utilise un système prototype de multi-fenêtrage, appellé APOTRE, développé au CNET, dans le cadre du projet CONCERTO de développement d'un poste de travail scientifique.

Actuellement, ce système ne possède pas de bibliothèque graphique. Cette absence nous a empéché de développer un prototype KABRI opérationnel.

Le programme utilise donc un bibliothèque de programmes graphiques, écrit en assembleur 68000, n'ayant aucune relation avec le système de multifenêtrage. Ce programme d'essai est écrit en C. Il n'a pour but que de tester certains aspects de l'interface.

Mais, nous avons choisi LISP, plus précisement le système Le_Lisp de l'INRIA [Chai], pour le futur dévellopement de KABRI. Pourquoi? Plusieurs éléments de réponse se trouvent dans le paragraphe suivant.

6. POUR UNE VERSION LISP DE KABRI

LISP a été mis au point par J. Mac CARTHY, dans les années 1958-1960, c'est à dire quand les langages s'appelaient FORTAN II, et que ALGOL sortait à peine de la phase de spécification.

LISP a, depuis, continué à exister sans modification et a profité de vingt ans d'usage de la part d'une partie des informaticiens, particulièrement des chercheurs en Intelligence Artificielle. Ceux-ci cherchaient la plus grande souplesse et puissance d'expression possible dans les manipulations symboliques, que ne possédaient pas les langages du type FORTRAN. Mais rien n'empêche de le sortir de ce cadre. Les systèmes LISP actuellememt disponibles sont devenus des environnements de programmation très puissants, comme INTERLISP pour ne citer qu'un exemple.

LISP possède les avantages de la plupart des autres langages comme PASCAL ou C:

- outils de mise au point de programmes

LISP fournit des outils de suivi d'exécution de fonction, ainsi que des outils de "déverminage" symbolique et dynamique (on arrête l'exécution, sur une erreur par exemple, on examine toutes les variables grâce à leur nom symbolique, on les modifie au besoin et on repart dans l'exécution à partir du point d'arrêt).

- fonctions d'accès au noyau du système

En l'occurence, les systèmes LISP permettent d'accèder en général, à toutes les fonctionnalités disponibles du noyau du système d'exploitation sur lequel ils ont été implantés: création de processus, communication entre processus, envoi de signaux, entrée-sortie de bas niveau, gestion des terminaux spécialisés comme les bitmaps.

LISP offre, en plus, tous les avantages des langages interprétés plus quelques autres que nous allons énumérer. Il possède, bien entendu des défauts que nous aborderons à la fin de ce paragraphe.

Mais voyons d'abord ses qualités:

- facilité de mise au point des programmes.

Comme tous les langages présenté sous forme conversationnelle, LISP permet le passage instantané entre le mode exécution et le mode édition.

De plus, les systèmes LISP comprennent en général des environnements puissants, comme le fameux DWIM de INTERLISP qui est un programme de correction automatique d'erreurs (Do What I Mean), ou CEYX de Le_Lisp qui permet une programmation de type objets et donnant la possibilité de définir des structures de plus en plus complexes sous forme de classes et sous-classes d'autres structures, et ceci d'une manière proche de la définition des types abstraits.

- traitement des erreurs par le programmeur.

En FORTRAN, PASCAL ou C, le programmeur doit lui-même vérifier l'entière validité des données, sous peine d'erreur fatale pour le programme.

LISP propose une approche différente. Le programmeur laisse le système LISP détecter lui-même l'erreur (de manière plus efficace), qui ne sera plus fatale. Cette erreur déclenche alors une fonction choisie suivant le type d'erreur parmi une liste de fonctions de traitement d'erreurs décrite

par le programmeur.

Prenons un exemple. Plutôt que de vérifier systématiquement que les arguments d'une fonction sont bien numériques, on laisse l'erreur se produire. LISP déclenche alors une fonction écrite par le programmeur lié à l'erreur "argument non numérique". Suivant le type de l'argument passé et la procédure où s'est produite l'erreur, on pourra remplacer cet argument par un autre licite et reprendre l'instruction interrompue, ou déclencher une nouvelle erreur qui sera alors traitée à un niveau supérieur.

Ces facilités sont indispensables dans la réalisation d'un logiciel interactif, aussi bien pour traiter les erreurs de l'utilisateur que pour des commandes qui doivent être accessible à n'importe quel moment (comme la commande pour sortir du système).

- possibilité de redéfinir les fonctionnalités de LISP lui-même

Parce que LISP est défini à partir du noyau très simple que sont les paires pointées et les fonctions de création et d'accès à celles-ci (les définitions peuvent être trouvées dans un ouvrage sur LISP, par exemple [Gira]) et à partir de LISP lui-même, on peut redéfinir toutes les fonctionnalité du langage, même la fonction EVAL qui est le coeur de l'interprète. Ceci est impossible en PASCAL ou C, à moins de modifier le compilateur.

- donnés et programmes sont identiques.

Données et programmes sont représentés sous la même forme: les S-expressions (S pour Symbolique), d'où la possibilité de faire manipuler des programmes LISP par un programme LISP.

- absence de déclaration des types de variables

Dans un langage compilé, le type de chaque variable ou structure doit être spécifié dans tous les modules l'utilisant. Chaque modification devient alors de plus en plus compliquée à effectuer.

En LISP, toute variable peut possèder quel type et la plupart des fonctions sont génériques. On permet ainsi une plus grande souplesse dans la modification de programmes.

- gestion des variables

Contrairement aux langages compilés, dans lesquels le nom des variables est perdu au moment de l'exécution, LISP les mémorise et permet ainsi la modification dynamique des variables globales pour s'adapter à l'utilisateur ou à l'environnement physique.

Dans les langages traditionnels, une variable n'est connue que dans la construction syntaxique où elle a été déclarée.

Lisp emploie la liaison dynamique de variables. Donnons un exemple. Soit deux fonctions

```
(de A (x) b)

(de B () (+ x 2))
```

Supposons l'existence d'une variable globale x de valeur 1. Si la fonction A est appellé avec un paramètre valant 5, elle rendra la valeur 7. L'appel de la fonction a caché la valeur de x comme variable globale la lie à la valeur 5. Pour B, x vaut toujours 5 et non 1. Ce n'est qu'à la fin de A que x retouvera sa valeur 1.

Ceci est totalement interdit avec PASCAL, par exemple. Pourtant cette méthode de liaison dynamique a des avantages.

Soit une fonction A, qui lie la variable VAR, appelle la fonction B, qui elle-même appelle la fonction C utilisant VAR. Dans un langage compilé, A devra passer la valeur de VAR comme argument à B, ainsi que B à C. Si la variable VAR faisait partie d'une modification de A et C, on est alors obligé de modifier B, et de récrire tous les appels à B.

En LISP, la fonction B ignorera VAR, mais C y aura accès, tous cela sans modification d'autres fonctions que A et C.

LISP est donc un des langages idéaux pour développer des prototypes comme le sont en général les systèmes interactifs, ou des logiciels dont le coût de développement est largement supérieur au coût d'utilisation. Ce point est important, si l'on note, comme J.BENTLEY dans Communication of the ACM de février 1985, qu'un programmeur produit en moyenne 2000 lignes de programme justes et documentées par an quelque soit

le langage employé.

Mais rien n'étant parfait, les problèmes de LISP viennent essentiellement des temps d'exécution et de l'encombrement mémoire, problèmes plus sensibles qu'en PASCAL ou C.

Les temps d'exécution peuvent, en partie, être réduit par l'utilisation d'un compilateur LISP, comme celui de Le_Lisp (avec un gain de 10 à 20). Mais ils restent supérieur à ceux d'un langage compilé.

L'encombrement mémoire est un problème plus délicat à régler. LISP offre des possibilités de modification physique des contenus des structures de données, mais avec tous les problèmes que cela peut poser.

Le dernier problème, et non le moindre, est celui de la portabilité entre systèmes LISP. Malgré la simplicité du LISP "pur et dur", qui permet en théorie de résoudre tous les problèmes, les différents systèmes LISP offrent heureusement de nombreuses (plus de 800 pour INTERLISP) fonctions plus puissantes. Malheureusement, toutes ces fonctions diffèrent d'un système à l'autre, ce qui rend difficilement portable un programme LISP. Heureusement, on trouve en général des fonctionnalités équivalentes (au nom et nombre d'arguments près).

On peut noter d'autre part, que la portabilité de langages plus traditionnel tel FORTRAN ou PASCAL, est loin d'être évidente (voir à ce propos, les réflexions de Y.GUIDO dans sa thèse [Guid]).

On pourra consulter le livre de J.J Girardot [Gira] qui constitue une bonne introduction à LISP.

Notre but étant de construire un logiciel par expérimentation, notre choix s'est donc porté sur LISP. Nous y voyons, en outre, l'intérêt de mener une expérience de programmation d'un autre type.

CHAPITRE 3

CONCLUSIONS

Nous avons décrit dans cette deuxième partie la spécification d'un logiciel interactif, en l'occurence KABRI.

De cette réflexion sur l'interactivité, nous avons tirés certains enseignements :

- une méthodologie de développement de l'interface de communication homme-machine à partir d'une analyse des tâches à accomplir.

- décomposition modulaire garantissant l'indépendance fonctionnalité, interface de communication, gestion des périphériques, permettant un modification rapide du logiciel pour l'adapter aux désirs de l'utilisateur.

Le développement d'un logiciel, durant sa période de vie, ne dépend pas uniquement du créateur, mais aussi des utilisateurs. Nous avons donc proposé une solution à l'extensibilité par la création de processus communicants.

Ces idées sont générales et nous espérons les utiliser dans la suite du développement de KABRI, ainsi que sur d'autres logiciels, comme le projet INGRES [Habi]. INGRES sera un poste de travail pour la planification de réseaux de télécommunication. Ce logiciel devrait servir aux planificateurs dans les régions (DTRN,..), aux gestionnaire de réseaux publics (TRANSPAC,...) ou privés (banques, grosses entreprise,..).

ANNEXE 1

COMMUNICATIONS ENTRE PROCESSUS AVEC UNIX

Les trois primitives FORK, EXEC et PIPE servent à créer et à faire communiquer des processus entre eux. Nous donnons dans cette annexe quelques explications sur ces mécanismes. Pour avoir des renseignements plus complets, se rapporter à une documentation UNIX par exemple [Unix].

Soit le processus P, représentant l'exécution d'un programme p.

Supposons qu'au moment où nous le considérons, il soit en train d'exécuter un appel à la primitive FORK.

Cette primitive système a pour effet de dupliquer intégralement le processus P, qui sera dorénavant le père, en un nouveau processus fils F indépendant. Les données, le code, le compteur ordinal et les descripteur des fichiers ouverts du père sont recopiés pour le processus fils.

La seule différence entre les deux processus est la valeur rendue par la primitive FORK:

- le fils reçoit la valeur 0
- le père reçoit une valeur différente de 0

Si l'on s'en tient là, les deux processus vont continuer de s'exécuter en parallèle avec le même code et les mêmes données, à partir de l'intruction suivant l'appel à FORK.

C'est là qu'intervient la primitive EXEC

Si l'on veut que dans le fils se déroule un programme différent de celui du père, On peut bien sûr faire un test du type :

```
si fork()=0 alors { programme du fils }
            sinon { programme du père }
```

mais on peut aussi faire appel à la primitive EXEC. Cette commande aura pour effet d'écraser le code et les données du processus fils par ceux du programme passé en paramètre. Le contrôle d'exécution est alors passé au point d'entrée du nouveau programme. Seuls les fichiers ouverts par le processus père restent accessibles à ce processus fils.

La primitive PIPE permet, elle, d'établir des "tubes" de communication entre processus d'une même famille (au sens de FORK). Les processus peuvent réaliser sur un tube les mêmes opérations de lecture et d'écriture que sur les fichiers.

PIPE, lors de son appel, crée une entrée en lecture et en écriture sur un tube. Lorsque qu'un programme "forke" (fourche), on obtient deux processus pouvant lire et écrire sur le même tube, donc pouvant communiquer. Il faut remarquer que la primitive EXEC préserve les accès aux fichiers ouverts, et donc au tube de communication. De plus, un tube peut être partagé entre plus de deux processus.

Un message écrit dans un tube par un seul ordre d'écriture est assuré de ne pas être découpé par d'autres messages écrits au même instant par d'autres processus connectés au même tube.

Ces communications sont du type bloquant (écriture sur un tube plein, lecture sur un tube vide).

UNIX propose un autre moyen de communication basé sur les signaux d'interruption envoyés par la primitive KILL. Ce signal est recu par le processus indiqué en paramètre et par tous ces fils.

La primitive SIGNAL permet à un processus d'indiquer au sytème quel sera sont comportement à l'arrivée d'un certain signal (l'ignorer, le traiter en indiquant une procédure de gestion, se suicider).

Un fils créé par FORK hérite des traitements des signaux de son père. Lors d'un EXEC, le nouveau processus ne posséde plus les procédure de gestion des signaux (écrasement du code).

ANNEXE 2

BIBLIOGRAPHIE SUR L'INTERACTIVITE

Une modélisation de l'interactivité graphique est proposée:

[Anso] E. ANSO, The device model of interaction, Computer Graphics, Vol.16, N°3, July 1982, pp.107-114.

Un article intéressant sur les méthodes de reprise:

[Arch] J.E.Jr ACHER, R. CONWAY, F. B. SCHNEIDER, User Recovery and Reversal in Interactive System, ACM Trans. on Programming Languages and System, Vol.6, N°1, January 1984, pp.1-19.

Une référence sur les environnements de programmation interactif:

[Bars] D.R. BARSTOW, H.E. SHROBE, E. SANDEWALL, Interactive Programming environment, McGraw-Hill, NY, USA, 1984.

Il s'agit d'un outil d'aide au programmeur, une de ses caractéristiques est la représentation multiple (texte et graphique) des objets manipulés:

[Bea1] M. BEAUDOIN-LAFON, C. GRESSE, Caty: un environnement

de programmation pour une construction graphique et interactive de programmes, TSI, Vol.3, N°4, Juillet-Août 1984, pp.261-271.

"PéTriPote: un prototype pour une application spécifique.", PéTriPote est un éditeur graphique de réseaux de Pétri:

[Bea2] M. BEAUDOIN-LAFON, version préliminaire communiquée par l'auteur, Thèse de Docteur Ingenieur, L.R.I, Faculté d'Orsay,1984.

L'étude des sorties qu'un programme graphique devrait avoir:

[Bent] J. BENTLEY, Programming Pearls : Graphic Output, Comm. of ACM, Vol.27, N°6, June 1984, pp.529-536.

Le système XS1 est un très bon exemple d'un système d'exploitation pensé pour l'interactivité:

[Bere] G. BERETTA, H. BURKHART, P. FINK, J. NIEVERGELT, J. STELOVSKY, H. SUGAYA, A. VENTURA, J. WEYDERT, XS1: An integrated interactive system and its kernel, Proc. 6th Int. Conf. Software Enginering, Tokyo, IEEE Computer Society, 1982, pp.340-349.

Dans ces deux livres, on trouvera une introduction à l'utilisation intelligente des formes, couleurs et relations dans un diagramme:

[Ber1] J. BERTIN, La graphique et le traitement graphique de l'information, Flammarion, Paris, 1977.

[Ber2] J. BERTIN, Sémiologie graphique, Mouton Gauthier-Villars, Paris, 1977.

CABRI, réalisé à Montpellier, est un logiciel interactif de manipulation de graphes:

[Bord] J.P. BORDAT, A. CAZES, M. CHEIN, O. COGIS, Y. GUIDO, CABRI: Première maquette d'un cahier de brouillon informatisé pour l'étude des ensembles ordonnés, Rapport de recherche N°5, CRIM ERA CNRS N°1045.

Un bon article sur les défauts fondamentaux de UNIX:

[Broz] H. BROZE, A. BRUFFAERTS, M.F. DECLERFAYT, E. HENIN, Ph. JAMART, M. LOBELLE, E. MILGROM, P. VERBATEN,

Y.D. WILLEMS, Evaluation critique du système UNIX, T.S.I., Vol.1, N°4, 1982, pp.315-324.

Cet article décrit un logiciel qui permet à un utilisateur de définir simplement ses propres menus:

[Buxt] W. BUXTON, M.R. LAMB, D. SHERMAN, K.C. SMITH, Towards a comprehensive user interface management system, Computer Graphics, Vol.17, N°3, July 1983, pp.35-42.

Documentation liée au projet CONCERTO:

[Cany] G.D. CANY, P.A. SIMONDON, Virtualiseur CONCERTO spécifications externes, Documents CNET, juin 1983.

Comparaison d'éfficacité entre plusieurs peripheriques interactifs.

[Car1] S.K. CARD, W.K. ENGLISH, B. BURR, Evaluation of mouse, rate-controlled isometric joystick, step keys and text keys for text selection on a CRT, Xerox Palo Alto Research Center, SSL-77-1, April 1977.

Cet article propose un modèle pour prédire les performances d'un utilisateur dans la réalisation d'une tache par ordinateur:

[Car2] S.K.CARD, T.P.MORAN, A.NEWELL, The Keystroke-Level Model for User Performance Time with Interactive System, C. of ACM, Vol.23, N°7, July 1980, pp.396-410.

[Car3] S.K. CARD, T.P. MORAN, A. NEWELL, The psychology of Human-Computer interaction, Lawrence Erlbaum Associates, 1983.

Le manuel de Le_Lisp pour SM90.

[Chai] J. CHAILLOUX, Le manuel de référece, Le_Lisp de l'INRIA, version 14, novembre 1984.

La planification des réseaux et le graphique dans une application développée au CNET Paris A:

[Coli] M.F. COLINAS, A computer graphic system for network planning, in Proc. of the second International

Network Planning Symposium, IEEE Conf. Publi N.215, London 1983, pp.199-204.

Documents de travail sur le multifenêtrage développé dans le cadre du projet CONCERTO:

[Con1] A. CONCHON, Environnement de programmation graphique interactif, Journées d'études CONCERTO, 16-17 décembre 1982, Perros-Guirec.

[Con2] A. CONCHON, Le poste de travail CONCERTO, Documents CNET/Paris A, mars 1984.

L'enseignement assisté par ordinateur est un des domaines où l'interactivité a été le plus étudiée:

[Couv] A. COUVERT, R. PEDRONO, AERO, un système d'Aide à l'Enseignement et à la résolution de problèmes en Recherche Operationnelle, T.S.I., Vol.2, N°2, 1983, pp.95-101.

Un autre projet de psote de travail pour la théorie des graphes :

[Cvet] D. CVETKOVIC, A project for using Computer in further development of graph theory, Proc. of the 4th. Conf. on theory and application of graph, Kalamazoo, 1980.

Le noyau graphique de SMALLTALK-80 sur SM90 réalisé au CNET Lannion A:

[Daus] P. DAUSSY, B. POTTIER, Le noyau graphique de SMALLTALK-80, BIGRE, N°39, juin 1984.

Ce livre contient un ensemble de rapport de recherche sur les systèmes interactifs en terme de langage de programmation, de matériel et de technique de dialogue:

[Dega] P. DEGANO, E. SANDEWALL, Integrated interactive computing systems, Proc. of the european conference on Integrated Interactive Computing Systems, ECICS 82, North-Holland Publishing Company, Stresa, Italy, 1-3 september, 1982.

Cet article décrit AGILE, langage pour la manipula-

tion interactive de graphes:

[Delg] J.P. DELGRANGE, A graph-theoretic language extension for an interactive computer graphics environment, Comput. & Graphics, Vol.5, 1980, pp.13-22.

La psychologie et la pédagogie peuvent aider à la conception d'une interface "amicale" envers l'utilisateur:

[Dwye] B. DWYER, A user-friendly algorithm, Comm. of ACM, Vol.24, N°9, September 1981, pp.556-561.

Une bonne introduction au développement de systèmes graphiques, malgré quelques faiblesses sur la partie consacrée aux problèmes liés à l'interactivité:

[Gard] Y. GARDAN, M. LUCAS, Techniques graphiques interactives et CAO, Hermes, Paris, 1983.

Ce livre représente un exposé assez complet du langage LISP:

[Gira] J.J. GIRARDOT, Les langages et les systèmes LISP une introduction, édiTESTS, 1985.

Une description de la norme graphique:

[Gks] ISO/DIS 1942, Graphical Kernel System, Functional description, 14 Nov.1982.

Smalltalk-80 est un langage et un système orienté objet développé à Xerox PARC:

[Gold] A. GOLDBERG, D. ROBSON, Smalltalk-80: The language and its implémentation, Addison-Wesley, 1983.

Une tentative de méthode pour spécifier les interfaces graphiques.

[Gree] M. GREEN, A methodology for the specification of graphical user interface, Computer Graphics, Vol.15, N°3, July 1981, pp.115-124.

Ces actes d'une conférence contiennent de bons articles sur l'interactivité:

[Gued] R. A. GUEDJ, P.J.W ten HAGEN, F.R.A HOPGOOD, H.A

TUCKER, D.A. DUCE, Methodology of interaction, IFIP Workshop on Methodology of interaction, Seillac, North-Holland Publishing Company, 1980.

Cf. [Bord]:

[Guid] Y. GUIDO, CABRI: un cahier de brouillon informatisé pour l'étude de la théorie des graphes, Thèse de 3ème cycle, Centre de recherche en informatique de Montpellier, Mars 1984.

Il s'agit de la description d'un éditeur et formateur de texte moderne:

[Gutk] J. GUTKNECHT, W. WINIGER, Andra: The Document Preparation System of the Personal Workstation Lilith, Software Practice and Experience, Vol.14, 1984, pp.73-100.

Le premier document sur INGRES:

[Habi] M. HABIB, J.P. RICHARD, Interactivite Graphique et Planification, Document d'Etude, CNET, DE/ATR/37-83, Novembre 1983.

Un thèse récente sur de nouveaux algorithmes de graphes:

[Hamr] M. HAMROUN, Sur quelques applications du parcours en profondeur dans les graphes, Thèse de troisième cycle, Université des Sciences et Techniques du Languedoc, Juin 1985.

Cet article décrit la notion d'acteur telle qu'elle est vue au MIT:

[Hewi] C. HEWITT, Viewing Control Structure as Patterns of Passing Messages, A.I. Journal, Vol.8, N°3, June 1977, pp.323-364.

Ce rapport décrit un logiciel de mise en page de livre:

[Irby] C. IRBY, L. BERGSTEINSSON, T. MORAN, W. NEWMAN, L.TESLER, A methodology for user interface design, rapport de recherche XEROX, PARC, Palo-Alto, janvier 1977.

Actes de la conférence de l'ACM Special Interest Group On Computer & Human Interaction (SIGCHI):

[Jand] A. JANDA, Human Factors in Computing Systems, Proc. of the CHI'83 Conference, Boston, Mass., 12-15 December 1983, North-Holland Publishing Company, 1984.

Le document décrivant un prototype de cahier de brouillon sur SM90 et sous UNIX:

[Kato] S. KATOSSKY, B. SEZNEC, Un cahier de brouillon sur SM90, Mémoire de Travail Personnel d'Option, Ecole des Mines de St-Etienne, juillet 1984.

La référence sur le langage C:

[Kern] B.W. KERNIGHAN, D. RITCHIE. The C programming language, Prentice-Hall Software Series, 1978.

Une application des langages orientés objet dans la simulation:

[Klah] P. KLAHR, D. McARTHUR, S. NARIAN, SWIRL: An Object-Oriented Air Battle Simulator, Proc. AAAI-82, Pittsburgh, August 1982.

Ce texte est un annexe au rappaort de la commission présidée par M.J. Yoccoz consacrée aux besions de la recherche en matière d'équipements informatique.

[Laza] D. LAZARD, Les besoins de la recherche mathématique en moyens informatiques, Gazette des Mathématiciens, Société Mathématique de France, N°27, Mars 1985.

Les auteurs décrivent dans ce livre, l'élaboration et l'expérimentation d'un assistant à la programmation tenant compte plus de l'utilisateur que des programmes:

[Ledg] H. LEDGARD, A. SINGER, J. WHITESIDE, Directions in human factors for interactive systems, Lecture Notes in computer Sciences, N°103, 1981, Springer Verlag ed.

Le développement du poste de travail STAR à Xerox PARC repose sur la notion d'acteur:

[Lipk] D.E. LIPKIE, S.R. EVANS, J.K. NEWLIN, R.L. WEISSMAN, Star Graphics: An Object-Oriented Implementation, ACM computer graphics, Vol.16, N°3, July 1982.

Cas typique du livre traitant uniquement du graphique et pas de l'interactivité malgrés son titre:

[Luca] M. LUCAS, La réalisation des logiciels graphiques interactifs, CEA-EDF-INRIA, Ecole d'été d'informatique, Eyrolles, Juillet 1979.

Cette thèse essaye de formaliser sous forme de types abstraits les langages graphiques et l'interactivité:

[Mall] W.R. MALLGREN, Formal specification of interactive graphics programming languages, An ACM distinguished dissertation 1982, The MIT Press.

Ces 6 articles traitent d'un langage de description de réseaux (NEDLAN) intéressant car il permet à l'utilisateur de définir son réseau avec une syntaxe à la BNF:

[Mar1] N. MAROVAC, A single data-display structure: A new view on interactive computer graphics in CAD, The Computer Journal, Vol.16, N°2, May 1973, pp.152-156.

[Mar2] N. MAROVAC, A method for defining general network for CAD, using interactive computer graphics, The Computer Journal, Vol.17, 1974, pp.332-336.

[Mar3] N. MAROVAC, The structures of the network definition language NEDLAN. Its use in defining networks in CAD using interactive computer graphics, Comput. & Graphics, Vol.2, 1976, pp.23-39.

[Mar4] N. MAROVAC, W. S. ELLIOT, A network-oriented language - a new approach to network design using interactives graphics, Comput. & Graphics, Vol.2, 1977, pp.235-239.

[Mar5] N. MAROVAC, A network oriented information structure: Networks semantics and structure, Comput. & Graphics, Vol.5, 1980, pp.41-52.

[Mar6] N. MAROVAC, W.S. ELLIOT, Interactive computer aided design using computer graphics, Comput. & Graphics, Vol.7, 1983, pp.177-187.

Ces écrits contiennent la documentation du fameux

logiciel CREDO:

[Mep1] J.M. MEPUIS, CREDO: un configurateur de réseaux de données, L'écho des recherches, N.110, Octobre 1982.

[Mep2] J.M. MEPUIS, CREDO-version CR1, Configurateur de réseaux de données, CNET, Tome 1: Note technique NT/PAA/ATR/RIP/929, Mars 1983.

[Mep3] M.LESK, J.M. MEPUIS, J.P. RICHARD, CREDO-version CR1, Configurateur de réseaux de données, CNET, Tome 2: Note technique NT/PAA/ATR/RIP/873, Décembre 1982 ; Tome 3: Note technique NT/PAA/ATR/RIIP/930, Mars 1983.

Une étude sur les représentations graphiques des structures de données:

[Meye] B.A. MEYER, Incense: a system for displaying data structures, Computer Graphics, Vol.17, N°3, July 1983, pp.115-125.

Une des premières études sur l'intérêt de l'utilisation de l'interactivité pour la résolution de problèmes NP-complets:

[Mich] D. MICHIE, J.G. FLEMING, J.V. OLDFIELD, A comparison of heuristic, interactive, and unaided methods for solving a shortest-route problem, Machine Intelligence 3, edited by D. Michie, N.Y. American Elsevier Publishing Company Inc, 1968, Edinburgh University Press.

Il s'agit de la description d'un langage de spécification de l'aspect utilisateur de l'interface de logiciels interactifs, du point de vue linguistique, psychologique et constructif:

[Mora] T.P. MORAN, The Command Language Grammar: a representation for the user interface of interactive computer system, Int. J. Man-Machine Studies, n°15, 1981, pp.3-50.

Un classique et une référence:

[Newm] W. NEWMAN, R.F. SPROULL, Principles of Interactive Computer Graphics, 2nd ed., Mc Graw-Hill, New-York, 1979.

Une étude des réalisations de logiciels interactifs dans le domaine de l'enseignement assisté par ordi-

nateur:

[Niev] J. NIEVERGELT, A pragmatic introduction to course-ware design, IEEE Computer, Vol.13, N°9, Sep.1980, pp.7-21.

Une critique de l'interface homme-machine du système d'exploitation UNIX:

[Norm] D. A. NORMAN, The trouble with UNIX, Datamation, Nov.1981, pp.139-150.

Rapport sur la réalisation d'interfaces sur la SM90 muni d'un bitmap et de UNIX:

[Pamm] K.G. PAMMET, Mouse-driven menu interfaces for soft-ware tools on a bitmap unix system, Rapport de recherche N°310, INRIA, 1984.

Un article sur des algorithmes liés à la gestion de plusieurs fenêtres sur un écran bitmap:

[Pike] R. PIKE, Graphics in overlapping bitmap layers, Computer Graphics, Vol.17, N°3, July 1983, pp.331-356.

Reflexions d'un théoricien des graphes sur l'utili-sation des ordinateurs dans son travail:

[Read] R.C. READ, Some applications of computer in graph theory, Beineke, Wilson Edt, Academic Press, 1978, pp.417-444.

L'animation par ordinateur est proche des concepts des systèmes basés sur la notion d'acteur:

[Reyn] C.U. REYNOLDS, Computer Animation with Scripts and Actors, Proc. of ACM SIGGRAPH Conference, July 1982.

Les deux références suivantes présentent l'ensemble des principes utilisés pour construire l'interface de la machine STAR de Xerox, modèle de LISA et de MACINTOSH de Apple:

[Seyb] J. SEYBOLD, Xerox's "Star", The Seybold Report, Seybold Publication, Vol.10, N° 16, Media, Pennsylvania, 1981.

[Smit] D.C. SMITH, C. IRBY, R.KIMBALL, B. VERPLANK E. HARSLEM, Designing the Star User Interface, BYTE Magazine, Vol.7, N°4, April 1982.

Emacs est un éditeur de texte écrit en LISP, permettant ainsi l'adaptation à l'utilisateur:

[Stal] R.M. STALLMAN, EMACS: The extensible customizable, self-documenting display editor, MIT, AI Memo 519a, Cambridge, Mass., June 1979.

Un éditeur présentant les programmes sous une forme structurée par la syntaxe du langage de programmation employé:

[Teit] T. TEITELBAUM, T. REPS, The Cornell Program Synthesizer: A Syntax-directed Programming Environment, Comm. of ACM, Vol.24, N°9, Sept.1981, pp.563-573.

Une des meilleures réalisations dans le domaine des systèmes interactifs, décrit un environnement de programmation LISP:

[Tei1] W. TEITELMAN et al., Interlisp Reference Manual, Xerox Palo Alto Research Center, Dec. 1975.

[Tei2] W. TEITELMAN, A Display Oriented Programmer's Assistant, Xerox Palo Alto Research Center, March 1977.

Le fameux système d'exploitation:

[Unix] Unix Time-Sharing System, numéro spécial de la revue: The Bell System Technical Journal, Vol.57, N°6, Part.2, 1978.

Ce livre présente une collection d'articles interdisciplinaires, informaticiens, psychologue, ingénieurs :

[Veer] G.C. van der VEER, M.J. TAUBER, T.R.G. GREEN, P. GORNY, Readings on Cognitive Ergonomics - Mind and Computer, Proc. of the 2nd European Conference, Austria, September 1984, Springer-Verlag, Lecture Notes in Computer Science.

LEG est un langage extensible de programmation d'algorithme de graphes:

[Vero] D.VEROVE, Un langage extensible de programmation d'algorithme de graphes, Thèse de 3e cycle, Université de Bordeaux I, Mars 1979.

De l'utilité de plusieurs représentations du même objet:

[Wate] R. C. WATERS, Program editors should not abandon text oriented commands, ACM SIGPLAN Notices, Vol.17, N°7, July 1982.

L'interactivité comme mise en forme de l'imaginaire:

[Wei1] J.L. WEISSBERG, Une nouvelle ingéniérie de l'imaginaire, Terminal 19/84, N°16, Oct 1983.

[Wei2] J.L. WEISSBERG, L'interactivité comme rapport social, Terminal 19/84, N°19-20, mai-juin 1984.

La plus vielle référence sur la réalisation d'un logiciel interactif de manipulation de graphe:

[Wolf] M.S. WOLFBERG, An interactive graph theory system, Ph.D, University of Pennsylvania, 1969.

Deux thèses sur un langage de traitement de graphes, LANGRAF, pour l'évaluation des structures de données associées:

[Zege] F. ZEGEL, Structures de données pour un langage de traitement de graphes, Thèse 3ème cycle, Université Paris VI, Décembre 1975.

[Zine] J.J. ZINETTI, Définition d'un langage de traitement de graphes, Thèse 3ème cycle, Université Paris VI, Décembre 1975.

ANNEXE 3

PROJET CABRI

CABRI

Cahier de Brouillon Informatique

Un Outil d'aide à la recherche et à l'enseignement dans le domaine de la théorie des graphes et des ensembles ordonnés

Responsables

Michel Habib	(Département d'Informatique, Ecole des Mines de St Etienne)
Jean-Marie Laborde	(Lab. Structures Discrètes (IMAG) Grenoble).

Participants

C. Benzaken	Grenoble
N. Balachef	Grenoble
R. Bonnet	Lyon
J. Bordier	Grenoble
O. Botta	Lyon
A. Bouchet	Le Mans
M. Deléglise	Lyon
P. Duchet	Paris
J.L. Fouquet	Le Mans
A. Germa	Orsay
M. Guillerault	Grenoble
M.Habib	St Etienne
J.-M. Laborde	Grenoble
M. Mollard	Grenoble
B. Peroche	Orsay. St Etienne
M. Pouzet	Lyon
D. Tallot	St Etienne
N.H. Xuong.	Grenoble

Ils relèvent principalement des formations suivantes

-Laboratoire Structures Discrètes (IMAG) - UA 393
-Département d'Informatique de l'Ecole des Mines de St Etienne - UA 815
-Equipe Combinatoire (Paris) - ER 193
-Centre de Recherche en Informatique (Le Mans) - UA 1099
-Equipe Combinatoire (LRI) - UA 410

Motivations

Dans les domaine de la théorie des graphes ou des ensembles ordonnnés comme dans celles qui en font usage, il arrive très fréquemment que l'on ait recourt à la réalisation de "petits dessins" , formés essentiellement de sommets et d'arêtes; cela dans le but de les manipuler, de leur appliquer concrétement certaines transformations ne serait-ce que pour vérifier telle ou telle propriété, conforter ou infirmer une idée, une conjecture.
Malheureusement cette pratique menée sur un brouillon ordinaire souffre de **limitations**, dûes par exemple à la taille relativement limitée des exemples que l'on peut traiter à la main ou encore au fait que la vérification d'une propriété peut être fort longue (comme c'est la cas, par exemple, lors de l'examen d'une "criticalité"); enfin la vérification, souvent, n'est au mieux, que faiblement garantie.
L'idée est donc naturelle de vouloir élargir les possibilités d'exploration par la mise à disposition des chercheurs du domaine (et d'étudiants) d'un outil graphique où **le traditionnel cahier de brouillon serait remplacé par l'écran interactif d'un terminal associé à toute la puissance de calcul et de mémorisation de l'ordinateur.**

C'est la réalisation d'un tel outil, sensé être d'accès aussi facile que possible pour être utilisé, pour partie, par des non informaticiens, qui constitue l'essence du projet CABRI.

Quelques choix

Si l'on réfléchit à un tel projet son caractère ambitieux saute immédiatement aux yeux: en effet il met en jeu des compétences variées, tout d'abord dans le domaine propre des théories mathématiques concernées, essentiellement la théorie des graphes et celles des ensembles ordonnées. Mais se posent évidemment aussi au niveau informatique des problèmes importants et actuels :

-conception de logiciel de taille importante, non entièrement spécifié et à caractère expérimental et évolutif (**Génie Logiciel**);

-développement d'interfaces graphiques interactives prenant en compte des aspects ergonomiques (**Interface Homme-Machine**);

-construction progressive au fur et à mesure des sessions d'utilisation du système, de bases de données, accessibles par différents utilisateurs (**Bases de**

Données Relationnelles);

-maîtrise d'environnements informatiques spécifiques, matériels et logiciels (aspects Système).

Vu l'ampleur du projet et si l'on prend en considération le nombre relativement faible de "programmeurs" il apparaît comme évident que nous ne nous fixons par pour objectif la réalisation d'un véritable produit mais plutôt celle d'un prototype évolutif, lié à un grand nombre de problèmes de recherche.

Il apparaît donc aussi comme inévitable qu'un certain nombre de choix soient faits pour restreindre les champs de recherche posés par un tel projet, pour partager, au nivau des différents équipes y concourrant, les différents axes de recherche et de même la réalisation des prototypes...
Initialement le projet, né dans l'environnement de la RCP "**Ensembles ordonnés et applications**" s'était surtout développé dans l'environnement du **Centre de Recherche en Informatique de Montpellier**, en particulier au travers de la réalisation d'un première maquette sur micro-ordinateur destinée à tester la faisabilité de l'ensemble, mais aujourd'hui il est largement repris pour tout un ensemble d'équipes mentionnées en tête de ce document.

Grands axes de recherche

L'activité du projet **CABRI** se trouve aujourd'hui essentiellement concentré autour des équipes mentionnées qui ont toutes, compte tenu de la puissance actuelle des micro et mini, en vitesse comme en stockage, opté pour une solution sur ces derniers types de matériel. c'est ainsi que les différent matériels sur lesquels nous travaillons sont par exemple

la SM 90 qui permet d'envisager la réalisation d'un véritable poste de travail **CABRI**,

Macintosh qui présente l'avantage d'être une machine déjà assez puissante et très répandue dans les universités,

le BFM 86 qui représente une certaine forme de compatibilité avec les micros 8-16 bits genre IBM-PC,

le VAX qui allie une grande puisssance à de bonnes possibilités de communications (avec la SM 90 par exemple) via UNIX .

Parmi les problèmes que le groupe se propose d'aborder on peut relever

-l'étude d'un cahier de brouillon sous **UNIX** utilisant une SM 90 et un écran "bitmapé", avec en particulier la réalisation d'un poste de travail intégrant un traitement de texte scientifique (St Etienne);

-le développement des possibilités de communication, via **TRANSPAC**, entre les différentes équipes utilisatrices de **CABRI**, ou le développant, leur permettant de partager certaines bases de données;

-la définition d'un langage interprété ouvrant à l'utilisateur la possibilité d'

opérateurs puissants, recursifs ou itératifs (Grenoble); de ce point de vue l'expérience acquise, à Paris lors de la conception de LANGRAF [1], et à Bordeaux avec la définition du langage LEG [2], devrait se montrer profitable

-la réalisation d'un système d'aide à la démonstration dans le domaine de la combinatoire (St Etienne-Lyon)

-la mise au point d'un générateur aléatoire de certaines structures combinatoires paramétrées par des propriétés (Grenoble). Dans la même direction la réalisation d'un système de production des plus petits objets d'une classe non vide d'objets combinatoires

-le développement d'outils permettant de manipuler des représentations graphiques de réseaux (de communication, de processeurs, ...).

Applications

Elles se déduisent naturellement de ce qui précéde et on en touvera ci-dessous une explicitation

disponible à cours terme :

-un poste de travail pour théoricien des graphes ou des ensembles ordonnés;

-un outil d'aide à l'enseignement (niveau universitaire) dans les domaines de l'algorithmique et des théories déjà mentionnées; on peut noter que les maquettes logicielles existantes (SM 90, Macintosh) ont déjà été utilisées à des fins d'enseignement (Grenoble, St Etienne);

à plus long terme :

-la réalisation d'un système, déduit de **CABRI**, mais manipulant essentiellement les éléments d'une figure de géométrie de l'enseignement secondaire, avec deux objectifs:

-mettre à la disposition des chercheurs en didactique un outil d'investigation spécifique

-fournir, dans le cadre de l'enseignement secondaire, un outil nouveau pour l'enseignement de la géométrie, dont les performances seront incomparables à celles des seuls crayons, règles et compas.

À ce niveau il probablement intéressant de signaler

-l'idée de la réalisation d'une conférence répartie (de type COSAC) sur, et à plus long terme avec, **CABRI**

-la volonté d'entretenir des rapports étroits avec d'autres recherches du même genre de recherche à l'étranger, dans l'état de nos connaissances, essentiellement le projet **Darwin**, un système expert en combinatoire, développé à l'UQAM (Montréal) et le projet **GRAPH** de Belgrade.

Moyens demandés

Les membres du groupe disposent à l'heure actuelle d'un certain nombre de matériels dont une SM 90, une BFM 86, quelques Macintoshs; mais le dévelopement actuel du projet nécessite une évidente extension. Seraient en particulier nécessaires

-une mémoire d'image (BitMap) (Numelec EHR 90)	35 KF
-deux MacIntoshs	50 KF
-un disque dur 60 Moctets	60 KF.

Il nous serait d'un grand secours que le PRC Mathématiques et Informatique puisse nous aider à acquérir certains de ces matériels.

Le développement de notre projet nécessite aussi un certain nombre de rencontres, dont nous estimons le coût à environ 20x1000 F soit 20 KF. Là encore les moyens de nos formations respectives ne sont pas suffisants et un aide serait la bienvenue.

Bibliographie (chronologique)

[1] projet LANGRAF

-F. Zegel *Structures de données pour un langage de traitement de graphes*, Thèse de 3ème cycle ,1975 Paris VI

-J.-J. Zinetti *Définition d'un langage de Traitement de graphes* , Thèse de 3ème cycle, 1975 Paris VI

[2] projet LEG

-D. Verove *Un Langage Extensible de programmation d'algorithmes de Graphes* , Thèse de 3ème cycle, 1979 Bordeaux 1

[3] prototype CABRI

Y. Guido *C.A.B.R.I. Un Cahier de Brouillon Informatisé popu l'étude de la théories des graphes* , Thèse de 3ème cycle, 1984 Montpellier

[4] D. Cvetkovics, L. Kraus *"GRAPH" an expert system for the classification and the extension of the Knowledge in the field of Graph Theory*, 1983, Faculté de Génie Electrique, Université de Belgrade

[5] F. Jaeger & J.-M. Laborde *Interprétation de dessins de graphes* , soumis au congrès AFCET de Reconnaissance des Formes et Intelligence Artificielle

[6] D. Tallot *Quelques propositions pour la mise en œuvre d'algorithmes combinatoires* , Thèse de 3ème cycle, 1985 St Etienne

QUELQUES PROPOSITIONS POUR LA MISE EN OEUVRE D'ALGORITHMES COMBINATOIRES

ANNEE : 1985
NOM DE L'AUTEUR : Didier TALLOT

UNIVERSITE DES SCIENCES ET TECHNIQUES DU LANGUEDOC
(MONTPELLIER II)

n° d'ordre:

RESUME

Le travail exposé dans ce rapport est composé de deux parties.

La première partie est constituée de l'étude de deux problèmes de géométrie algorithmique:

- le calcul de l'enveloppe convexe d'un ensemble de points
- la méthode de balayage du plan

Dans la deuxième partie, nous nous intéressons à la réalisation d'un logiciel interactif et extensible de manipulation de graphes et d'ensembles ordonnés: KABRI. On y trouve la description de l'interface de ce logiciel ainsi que de son architecture.

Mots-clés:

Géométrie Algorithmique
Enveloppe Convexe
Balayage du Plan
Type Abstrait
Système interactif
Communication homme-machine
Facteur humain
Conception de systèmes
Algorithme sur les graphes