Application des graphes de programme à l'optimisation d'un modèle destiné à mesurer les performances d'un système d'exploitation

Martine Lepeuve

N° d'ordre

Thèse

Présentée à

L'UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

pour obtenir

le grade de Docteur de troisième cycle

PAR

Martine Lepeuve

APPLICATION DES GRAPHES DE PROGRAMME A L'OPTIMISATION D'UN MODELE DESTINE A MESURER LES PERFORMANCES D'UN SYSTEME D'EXPLOITATION

Thèse soutenue le 9 janvier 1973 devant la commission d'examen

Monsieur J. KUNTZMAN Président

Monsieur G. VEILLON Examinateur

Monsieur I . BOLLIET Examinateur

UNIVERSITE SCIENTIFIQUE

ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

Président : Monsieur Michel SOUTIF
Vice-Président : Monsieur Gabriel CAU

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du Solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHENE Marcel	Chimie papetière
	COEUR André	Pharmacie chimique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures

MM.	GASTINEL Noël	Analyse numérique
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physiologie végétale
MM.	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques Appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MASSEPORT Jean	Géographie
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du Solide
	OZENDA Paul	Botanique
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servomécanismes
	PILLET Emile	Physique industrielle
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	SANTON Lucien	Mécanique
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
Mme	VEYRET Germaine	Géographie
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	BULLEMER Bernhard	Physique
	RADHAKRISHNA Pidatala	Thermodynamique

PROFESSEURS SANS CHAIRE

MM.	AUBERT Guy	Physique
	BEAUDOING André	Pédiatrie
	BERTRANDIAS Jean-Paul	Mathématiques Appliquées
	BIARES Jean-Pierre	Mécanique
	BONNETAIN Lucien	Chimie minérale
Mme	BONNIER Jane	Chimie générale
MM.	CARLIER Georges	Biologie végétale
	COHEN Joseph	Electrotechnique
	COUMES André	Radioélectricité
	DEPASSEL Roger	Mécanique des Fluides
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DOLIQUE Jean-Michel	Physique des Plasmas
	GAUTHIER Yves	Sciences biologiques
	GEINDRE Michel	Electroradiologie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	JANIN Bernard	Géographie
Mme	KAHANE Josette	Physique
MM.	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	POULOUJADOFF Michel	Electrotechnique
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction Mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
M.	VALENTIN Jacques	Physique nucléaire

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques Appliquées
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BOUVARD Maurice	Mécanique des Fluides
	BRIERE Georges	Physique expérimentale
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse

MM.	CHIAVERINA Jean	Biologie appliquée (EFP)
	CHIBON Pierre	Biologie animale
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique
	CROUZET Guy	Radiologie
	DURAND Francis	Métallurgie
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GAVEND Michel	Pharmacologie
	GENSAC Pierre	Botanique
	GERMAIN Jean-Pierre	Mécanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques Appliquées
	GROULADE Joseph	Biochimie médicale
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Médecine préventive
	IDELMAN Simon	Physiologie animale
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	JOLY Jean-René	Mathématiques Pures
	JOUBERT Jean-Claude	Physique du Solide
	JULLIEN Pierre	Mathématiques Pures
	KAHANE André	Physique générale
	KUHN Gérard	Physique
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LAJZEROWICZ Joseph	Physique
	LANCIA Roland	Physique atomique
	LE JUNTER Noël	Electronique
	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique Nucléaire
	LONGEQUEUE Jean-Pierre	Physique Nucléaire
	LUU DUC Cuong	Chimie Organique
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et Médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)
	MAYNARD Roger	Physique du Solide
	MICOUD Max	Maladies infectieuses
	MOREAU René	Hydraulique (INP)
	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PERRIN Louis	Pathologie expérimentale
	PFISTER Jean-Claude	Physique du Solide
	PHELIP Xavier	Rhumatologie
Mle	PIERY Yvette	Biologie animale
MM.	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RICHARD Lucien	Botanique
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	ROUGEMONT (DE) Jacques	Neuro-Chirurgie
	STIEGLITZ Paul	Anesthésiologie

MM. STOEBNER Pierre	Anatomie pathologique
VAN CUTSEM Bernard	Mathématiques Appliquées
VEILLON Gérard	Mathématiques Appliquées (INP)
VIALON Pierre	Géologie
VOOG Robert	Médecine interne
VROUSSOS Constantin	Radiologie
ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

MM. BOUDOURIS Georges	Radioélectricité
CHEEKE John	Thermodynamique
GOLDSCHMIDT Hubert	Mathématiques
YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MATIRES DE CONFERENCES

Mme BERIEL Hélène	Physiologie
Mme RENAUDET Jacqueline	Microbiologie

Fait le 8 MARS 1972.

Je tiens à remercier

Monsieur J. KUNTZMANN de m'avoir fait l'honneur de bien vouloir présider mon jury.

Monsieur G. VEILLON de m'avoir si bien orientée et dirigée dans mes travaux.

Monsieur L. BOLLIET d'avoir accepter de faire partie de mon jury.

Monsieur M. BENNETT de m'avoir fourni mon sujet et de m'avoir accueillie dans son équipe.

Le secrétariat et les personnes du service tirage qui ont permis la réalisation matérielle de ce travail.

I - INTRODUCTION

Un moyen naturel de représenter un programme est de faire un organigramme, c'est-à-dire un graphe où les noeuds de formes géométriques variées figurent les différentes instructions et les arcs le transfert du contrôle de l'une à l'autre.

Il existe déjà beaucoup de modèles pour représenter sous forme de graphe, l'exécution d'un travail particulier en machine. Ces modèles ont fait l'objet de nombreuses études théoriques, nous pouvons citer, par exemple, les travaux de PETRI ou ceux de IANOV, (voir la bibliographie : [P1, I1]).

Dans les pages suivantes, nous allons utiliser une de ces représentations sous forme de graphe, pour résoudre un problème de minimisation. Ce travail entre dans le cadre d'un projet de mesure de performance d'un système réalisé par M. BENNETT du centre de recherche C.I.I. de Grenoble.

Pour mesurer efficacement les performances d'un système, il faut pouvoir faire varier le hardware sur lequel il se trouve implanté. Il est en effet intéressant de voir comment se comporte le système en question suivant la nature du matériel utilisé.

Pour celà, nous nous servons d'un simulateur de machine virtuelle. Le modèle mis en oeuvre peut être représenté par le schéma de la figure n° 1. Nous travaillons sur un ordinateur C.I.I. 10070 et sous le système SIRIS7 [S3]. Sont chargés en machine, le simulateur, le système à mesurer (ici SAM [S2]) ainsi que les programmes servant de données à ce dernier. Ces programmes constituent un échantillonnage assez varié et assez représentatif des traitements qui seront demandés

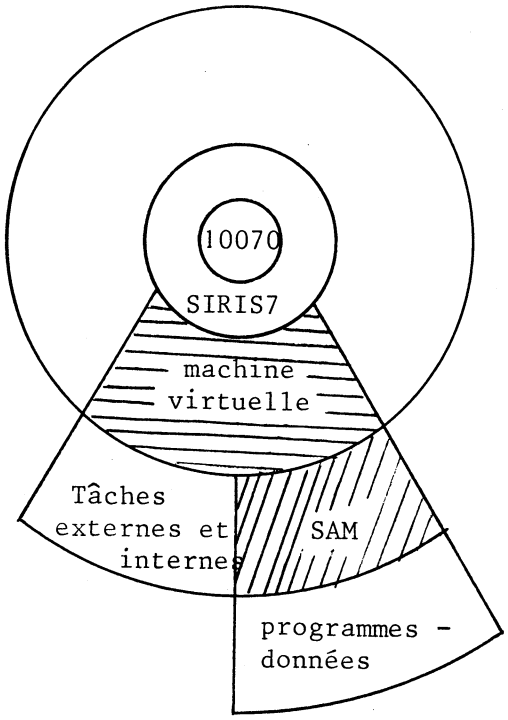


FIGURE n° 1

par les utilisateurs futurs du système, afin de valider les mesures qui seront prises.

SAM est un système de multiprogrammation [S2].

La multiprogrammation consiste en la coexistence de plusieurs programmes-utilisateurs dans le système. Le déroulement d'un de ces programmes peut être interrompu au profit d'un autre avec reprise ultérieure de celui-ci. Les différents travaux exécutés sur la machine virtuelle peuvent être divisés, pour la simulation, en trois catégories que nous appellerons respectivement : processus, tâches externes et tâches internes.

I.1. Les processus

Ce sont l'ensemble des travaux demandés par les programmes systèmes et utilisateurs à l'unité centrale. Leur déroulement se fait suivant une succession logique. Dans notre simulation, SAM et les programmes-données constituent un certain nombre de processus.

I.2. Les tâches externes

Ce sont les opérations qui se réalisent, comme par exemple, les entrées/sorties, à l'extérieur de l'unité centrale. Bien qu'elles soient

souvent activées par des processus travaillant sur l'unité centrale, nous pouvons considérer que leur déroulement se fait totalement de manière indépendante des autres travaux exécutés en machine.

Ces tâches externes peuvent provoquer des interruptions de l'unité centrale : comme par exemple lors d'une fin de lecture ou d'écriture.

I.3. Les tâches internes

Ce sont les séquences partiellement câblées qui réalisent le traitement immédiat des interruptions dûes aux tâches externes. Elles sont totalement indépendantes des processus qui sont en cours d'exécution.

Les tâches externes se font parallèlement aux autres travaux; elles utilisent d'autres unités de traitement que l'unité centrale : ce sont les unités périphériques. Les tâches internes et les processus se déroulent en simultanéïté : c'est-à-dire qu'à un instant donné bien qu'un seul de ces travaux soit en train de s'exécuter, plusieurs d'entre eux peuvent être en cours d'exécution et attendent pour poursuivre cette exécution que le contrôle leur soit à nouveau donné. Le passage de l'un à l'autre de ces travaux se fait au moyen de certaines tâches internes.

Simulons maintenant le fonctionnement parallèle de deux unités centrales que nous noterons UC1 et UC2. Tour à tour, nous avançons le déroulement de chacune d'elles d'un certain laps de temps. A un moment donné UC1 en est au temps t1 de son exécution et UC2 au temps t2 de la sienne ($t1 \neq t2$).

Si elles ne se transmettent aucune information, il n'est pas nécessaire de les synchroniser. En effet, elles travaillent alors de manière totalement indépendante.

Par contre, supposons maintenant qu'il existe un emplacement mémoire M où elles viennent lire et écrire à tour de rôle. Avant de lire ou d'écrire au temps t, pour l'exécution de l'une, nous devons nous assurer que toutes les lectures et écritures devant être réalisées avant t, pour le compte de l'autre, ont bien eu lieu.

Le fonctionnement des deux unités doit être ainsi synchronisé.

Le problème est sensiblement le même entre les processus et les autres travaux dont nous simulons les exécutions. Nous associons à chacun d'eux le temps réel où en est sa simulation.

. *Synchronisation entre processus et tâches externes*

Les tâches externes se réalisent sur d'autres unités que l'unité centrale. Nous sommes exactement dans le cas précédent où deux unités travaillent en parallèle.

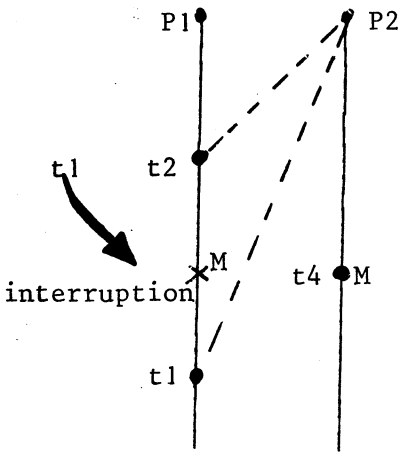
. *Synchronisation entre processus et tâches internes*

Les interruptions sont prévisibles à l'avance dans la simulation. Si au temps t de son exécution, un processus écrit ou lit dans un emplacement M, il faut être certain que toutes les interruptions prévues avant t, qui activent une tâche susceptible de lire ou d'écrire dans M, ont bien été simulées.

. *Synchronisation entre deux processus*

A la suite d'une interruption, ce n'est pas forcément le processus qui était en train de s'exécuter qui reprend le contrôle. Nous allons voir, sur un exemple, comment celà nous oblige à synchroniser les processus entre eux.

Supposons qu'une interruption prévue au temps t1 soit simulée alors que la simulation du processus P1 en est, elle, au temps t2 (t2 < t1).



Après l'interruption, nous commençons à interpréter le processus P2. Il lit ou écrit au temps t4 dans un emplacement M. Nous devons alors nous assurer que la simulation de P1 a été poursuivie jusqu'à la dernière utilisation de M précédent le temps t1. Sinon nous n'avons pas forcément dans M l'information qui devrait s'y trouver.

Les processus doivent donc être synchronisés avec les autres travaux à des endroits précis de leur exécution : avant chaque lecture ou écriture dans un emplacement utilisé par un autre processus, une tâche externe ou une tâche interne. Nous appellerons ces endroits, points de synchronisation et nous les noterons par le symbole ω. Entre deux de ces points, l'exécution d'un processus est indépendante des autres travaux en machine.

Le schéma de la figure n° 2 donne un exemple de ce qui se passe dans le temps pour une unité centrale et deux unités périphériques E1 et E2 qui fonctionnent en parallèle.

Chacune des périphériques exécute un certain nombre de tâches externes qui produisent certaines interruptions dans le déroulement de l'unité centrale. Ces interruptions ont été notées au moyen de traits verticaux. L'unité centrale traite une suite de séquences des processus et des tâches internes dont nous avons noté les passages par les points ω.

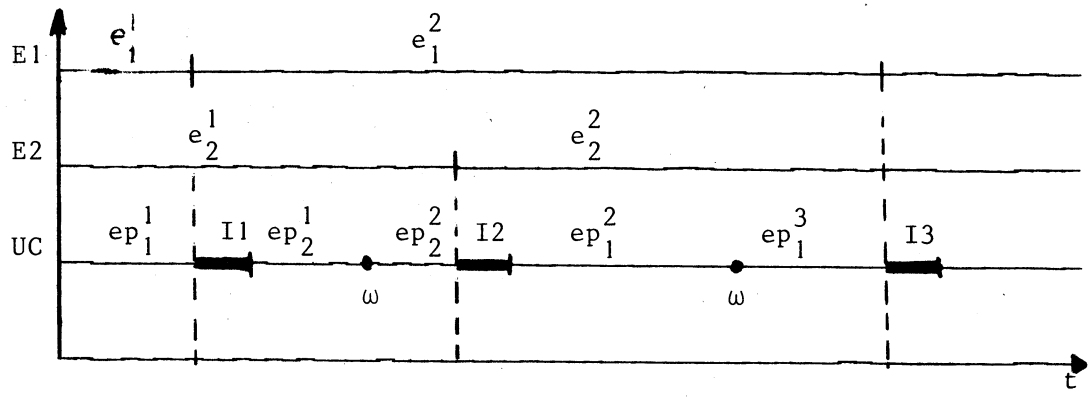


FIGURE n° 2

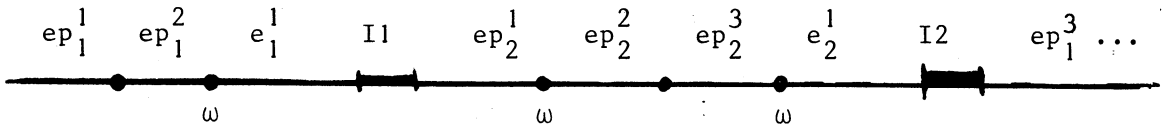
e_i^j représente la j^{em} séquence s'exécutant entre deux interruptions sur l'unité externe Ei

Ij c'est la j^{em} tâche interne s'exécutant après la j^{em} interruption de l'unité centrale

ep_i^j c'est la j^{em} séquence du processus p_i qui se trouve exécutée sur l'unité centrale.

Pour simplifier les choses, nous avons supposé que seulement deux processus s'exécutent simultanément et que chaque interruption fait passer de l'un à l'autre.

Pour tenir compte des nécessités de synchronisation précédemment exposées, la simulation des différentes séquences, exécutées sur les trois unités, se fera selon l'ordre du schéma suivant :



Chaque fois que nous simulons l'exécution d'un processus nous avançons jusqu'au prochain point ω. Deux cas peuvent alors se produire :

. Aucune interruption n'aurait dû se produire avant le temps t où on en est dans l'exécution du processus en question. Nous continuons alors sa simulation.

. Une ou plusieurs interruptions aurait dû se produire avant t. Nous faisons alors dérouler l'unité périphérique produisant la première interruption dans le temps. Puis nous simulons le traitement de l'unité centrale consécutif à cette interruption.

Un compteur de temps est associé à chaque processus, à l'unité centrale et aux deux périphériques. Notons Cp1 Cp2 Cuc CE1 et CE2 ces compteurs qui sont au départ tous à 0. Après l'exécution de ep_1^1 et ep_1^2 l'état des compteurs est le suivant : Cp1 = t1 Cp2 = 0 Cuc = t1 CE1 = CE2 = 0. Nous tenons alors compte de l'interruption qui aurait dû avoir lieu au temps t inférieur à t1. Cuc est ramené à t et nous réalisons la séquence e_1^1. Le processus P1 est en avance sur son exécution. Lorsque nous reprendrons cette exécution il faudra rajouter à Cp1 le temps d'unité centrale passé en dehors de P1. Cuc est alors avancé au temps où en est l'exécution de P1 : CUC = Cp1.

Nous pouvons voir qu'en fait l'ensemble des séquences réalisées pour l'unité centrale est le même que dans la réalité et que l'avance prise sur l'exécution des processus n'influe pas le comportement du système mesuré.

Chaque processus subit une exécution simulée sur la machine virtuelle. Les instructions privilégiées, par exemple, sont interprétées et correspondent à l'exécution d'une séquence d'instructions réalisant une modification de l'état de la machine virtuelle.

Le temps passé pour le compte de chaque instruction ne correspond pas à celui qu'elle aurait mise pour s'exécuter sur la machine en question.

Aussi nous simulerons leur temps d'exécution. Les compteurs qui fourniront au point de synchronisation le temps où en est réellement l'exécution, doivent être remis à jour par des appels à un sous-programme qui seront insérés dans le code des différents processus.

Représentons chacun de ces processus par un graphe dont les arcs correspondent à des suites d'instructions et les noeuds aux jonctions entre **elles**. Nous verrons plus loin comment une telle représentation peut être obtenue à partir d'un programme écrit dans un langage particulier. Une solution immédiate, pour la remise à jour des compteurs consiste à insérer sur chaque branche des appels à un sous-programme qui ajoutera le temps d'exécution de la branche en question, préalablement évalué à partir des instructions qui s'y trouvent.

Pour des programmes complexes, comme les programmes systèmes, le nombre des instructions par branche est assez réduit et en mettant un appel sur chacune d'elles nous pouvons obtenir jusqu'à un appel pour trois ou quatre instructions. Cela accroît énormément le temps d'exécution du modèle.

La séquence de mise à jour se trouve dans le simulateur de machine virtuelle. Nous y accèdons au moyen d'un SVC (SuperVisor Call) qui est interprété par le simulateur. Cela nécessite un certain nombre d'instructions de sauvegarde et de restauration.

De plus, pour ne pas perturber, s'il y a lieu, la pagination du système mesuré et par conséquent son fonctionnement, chaque appel rajouté remplace l'instruction machine qui vient immédiatement après dans le code généré. Cette instruction est gardée en mémoire, dans une table afin d'être exécuté avant le retour au processus appelant.

Celà correspond en tout à une douzaine d'instructions machines.

Le temps d'exécution du modèle se trouve alors multiplié par quatre ou cinq.

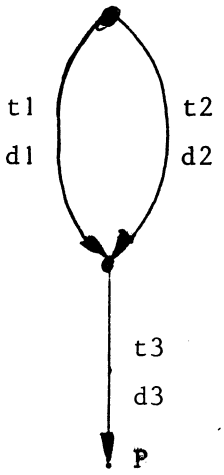


FIGURE N° 3

Soit le graphe de la figure n° 3 t1, t2, t3 sont les temps d'exécutions des arcs d1, d2 et d3.
Si nous mettons sur d1 un appel pour rajouter **au compteur** t1 - t2 et sur d3 un pour ajouter t3 + t2, nous obtenons toujours en arrivant à P la même valeur pour le temps d'exécution du chemin parcouru.
En effet, si ce chemin est d1 d3 la valeur obtenue est t1 - t2 + t3 + t2 soit t1 + t3.
Si c'est d2 d3 la valeur est t2 + t3.

Nous pouvons, comme le montre cet exemple, supprimer un certain nombre d'appels, en modifiant les valeurs ajoutées aux compteurs tout en obtenant bien le temps réel où en est l'exécution, aux points de synchronisation.

Le problème, que nous allons résoudre consiste à minimiser le nombre des appels au sous-programme de remise à jour, nécessaire pour avoir le temps exact où en est un processus donné lors de sa synchronisation avec les autres travaux.

Nous pouvons distinguer deux degrés dans cette minimisation :

. La minimisation en nombre : Elle consiste à minimiser le nombre d'appels insérés dans le code de chacun des processus. Nous verrons la similitude de ce problème avec celui de KNUTH [K2] et comment nous pouvons utiliser le théorème qu'il a énoncé sur la décomposition des chemins d'un graphe.

Nous verrons également comment la construction d'une telle solution peut se ramener à un problème classique de la théorie des graphes: la détermination d'un arbre maximal.

. La minimisation en moyenne : Elle consiste à minimiser le nombre d'appels en moyenne lors du déroulement de chaque processus. Nous tenons compte alors de la fréquentation des arcs à l'exécution.

Nous verrons, dans une première partie, quelles sont les solutions théoriques apportées à ces deux niveaux. Dans une seconde partie, nous étudierons l'application pratique de ces algorithmes à des programmes écrits dans un langage particulier : Lp 70.

Nous verrons les différents problèmes posés par la construction du graphe, sa représentation en machine et la programmation des algorithmes précédemment exposés.

Dans une dernière partie, nous comparerons les résultats obtenus à partir des algorithmes de minimisation en nombre et de minimisation en moyenne, relativement au temps d'exécution du modèle.

II - ETUDE GENERALE SUR LES GRAPHES DE CALCUL

On appelle graphe de calcul tout graphe qui représente l'exécution d'un certain travail sur un ordinateur. Ces graphes peuvent être divisés en deux catégories :

- les graphes de calcul séquentiel,
- les graphes de calcul parallèle.

II.1. Graphe de calcul séquentiel

Ce sont les graphes qui représentent un programme s'exécutant sur une machine à un seul processeur. Le contrôle est donné à une seule action à la fois. C'est le cas des organigrammes traditionnels, par exemple.

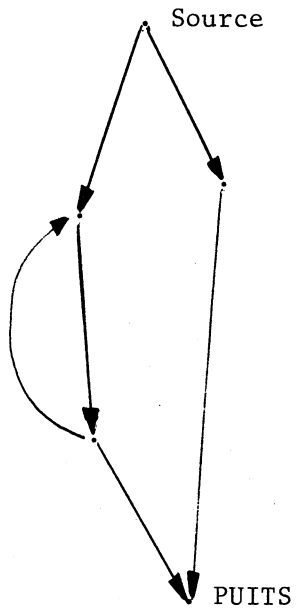


FIGURE N° 4

Ce sont soit les noeuds soit les arcs qui représentent les différentes instructions à exécuter. Dans les deux cas, le transfert se fait sur une seule branche à la fois.
Le graphe possède un seul noeud source et un seul noeud puits qui sont respectivement les noeuds de départ et d'arrivée des exécutions possibles.
Une exécution particulière peut être représentée par un chemin du graphe.
Parmi les différents chercheurs qui ont travaillé sur ce type de graphes, nous pouvons citer : IANOV qui a apporté un certain formalisme et s'est particulièrement intéressé aux problèmes d'équivalence [I1].

- COOPER et KNUTH qui se sont intéressés aux organigrammes dans le but d'étudier les performances des programmes [C1, K2].

- LOWE qui a cherché à appliquer l'étude de ces graphes aux problèmes de pagination [L1]

- MARTIN et ESTRIN qui ont fait des études statistiques sur leurs exécutions en rajoutant sur chaque arc la probabilité qu'il a d'être choisi à partir du noeud précédent[M1, M2].

Le but principal de toutes ces études est d'acquérir une meilleure connaissance des programmes et de la façon dont on peut les optimiser.

II.2. Graphe de calcul parallèle

L'introduction de calculateur à processeurs multiples a amené de nombreuses équipes de chercheurs à s'intéresser aux moyens de représenter le déroulement de programmes où plusieurs instructions s'exécutent en même temps. Il s'agit, en fait, de trouver un outil pour résoudre les problèmes dûs au parallèlisme :

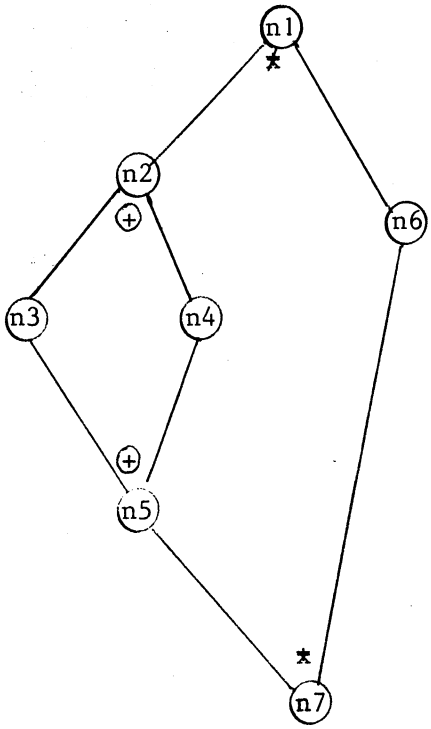
- synchronisation des lectures et écritures en mémoire,
- affectation des différents processeurs qui se trouvent en nombre limité
- évaluation de performance.

Il existe déjà de nombreux modèles de calcul parallèle. Parmi ceux-ci, nous citerons :

II.2.1. Les graphes bilogiques

Ces graphes ont fait l'objet de nombreuses publications de la part de G. ESTRIN, D. MARTIN, E. RUSSEL, R. TURN, J. BAER et D. BOVET (E1, M1, M2, R1, B2).

Le transfert du contrôle et l'activation du travail représenté par chaque noeud se fait au moyen de deux types de logique. Chaque noeud est muni de sa logique d'entrée et de sa logique de sortie.



\+ logique disjonctive

* logique conjonctive

FIGURE N° 5

. La logique disjonctive (OU)

Un noeud muni d'une logique d'entrée disjonctive ne s'active que lorsque le contrôle arrive par une seule des branches incidentes.

Après l'exécution d'un noeud muni d'une logique de sortie disjonctive, le transfert du contrôle se fait sur un seul des arcs qui en sortent.

. La logique de type conjonctive (ET)

Un noeud muni d'une logique d'entrée conjonctive ne sera activé que lorsque le transfert du contrôle se fait sur tous les arcs y arrivant.

Après l'exécution d'un noeud muni d'une logique de sortie conjonctive, le transfert du contrôle se fait sur tous les arcs qui en partent.

La principale difficulté que pose l'étude de ces graphes, consiste à déterminer et à représenter toutes les exécutions possibles dans le cas où il y a des circuits. Toutes les études statistiques, faites à leur sujet par MARTIN et ESTRIN [M1, M2] concernent uniquement les graphes sans circuits. Aussi ont-ils définis une série de transformations pour passer d'un graphe quelconque à un graphe sans circuit équivalent pour certain calcul statistique et notamment celui du temps moyen d'exécution à partir des probabilités des arcs [M3].

II.2.2. Les "computation graphs"

Ce modèle a été mis au point par KARP et MILLER [K3]. Chaque arc est le support d'une file de données et chaque noeud représente une opération qui lit ses données sur les arcs en entrée et écrit ses résultats sur ceux en sortie.

Soit dp un arc allant du noeud ni au noeud nj. Soient Oi et Oj les opérations associées à ni et nj. L'arc dp est décrit par quatre entiers qui sont :

Ap : le nombre de données initialement dans la queue
Up : le nombre de données rajoutées par Oi
Wp : le nombre de données prises par Oj
Tp : la longueur minimum de queue nécessaire à l'exécution de Oj.

Le fonctionnement du modèle est décrit par une suite S = (S1, ..., SN, ...) de sous-ensembles des opérations du graphe. SN représente l'ensemble des opérations exécutées en simultanéité au $N^{ème}$ pas.

La portée représentative de ces graphes est restreinte par le fait qu'ils excluent les branchements sur test.

Ils peuvent être utilisées, par exemple, pour l'étude des microprogrammes. La microprogrammation offre, en effet, un découpage du calcul en phases asynchrones qui comportent un certain nombre d'opérations simultanées.

II.2.3. Les "Flow Graph Shemata" (D. SLUTZ [S1])

Ils sont consitués par deux graphes :

- Le graphe de calcul qui indique les différentes opérations et leurs zones de lecture et d'écriture.

- Le graphe de contrôle qui détermine l'ordre dans lequel doivent être réalisées ces lectures et écritures.

Un tel modèle semble fournir un parallèlisme maximum, puisque seules les contraintes d'ordre sur les lectures et écritures servent à synchroniser les opérations.

II.2.4. Les réseaux de Petri [P1]

Ils ont pour origine les graphes de transition d'états dont ils diffèrent par la possibilité de parcourir plusieurs branches parallèlement et par l'adjonction de signaux qui permettront la synchronisation.

Ces réseaux sont à la base de la création de nombreux autres modèles comme ceux de Holt ou de Luconi.

Nous citerons également les travaux de Rodriguez-Bezoz, Van Horn et Muller.

De ces différentes études, nous pouvons dégager un certain nombre de problèmes généraux :

- le déterminisme du modèle : Un modèle est dit déterministe si et seulement si l'exécution ne dépend que des données initiales.
- l'équivalence de deux graphes : Deux graphes sont équivalents si pour les mêmes données initiales, ils donnent le même résultat.
- L'étude de leur performance dans le cas d'une implémentation en machine :
  - temps d'exécution
  - place mémoire utilisée.

Chacun de ces graphes peut avoir de par leur nature des applications propres : conception du Hardware, microprogrammation, etc...

II.3. Les graphes de programme

Nous allons étudier maintenant le graphe que nous utiliserons pour représenter chacun des processus.

Il s'agit d'un graphe de calcul séquentiel dont chaque arc correspond à une suite d'instructions et chaque noeud à une jonction entre certaine de ces suites.

Exemple :

Soit le programme Lp70 suivant : [L2]

```
BEGIN
R1:=0;
R2:=1;
LOOP BEGIN
R1:=R1+1
R2:=R2+R1;
IF R1=100 THEN GOTO A;
END
A:WRITE (R2);
END.
```

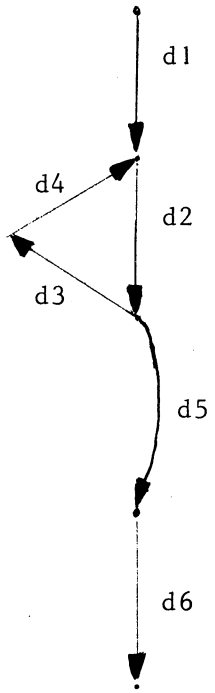


FIGURE N° 6

Il calcul la somme des cent premiers nombres entiers.
Nous pouvons lui faire correspondre le **graphe** de la figure n° 6 avec pour chaque arc les suites d'instructions suivantes :

d1 : R1:=0; R2:=1;
d2 : R1:=R1+1: R2:=R2+R1: le test R1=100
d3 : branchement après le IF dans le cas où le test est négatif
d4 : arc de retour de la boucle
d5 : GOTO A
d6 : WRITE(R2)

DEFINITION

Un graphe de programme est un doublet $(X,\mathcal{U})$ où X est l'ensemble des noeuds et $\mathcal{U}$ l'ensemble des arcs qui représentent chacun une liaison entre deux noeuds de X.

A chaque arc dp de $\mathcal{U}$ est associée une suite d'instructions Ip.

REMARQUES

. La définition du graphe par le doublet $(X,\mathcal{U})$ est semblable à la définition des réseaux orientés : [K4].

Les articulations sont les noeuds, et les couples de connections les arcs orientés.

Cette définition permet l'existence d'arcs parallèles.

Deux arcs sont dits parallèles s'ils ont même origine et même extrémité.

Par exemple, soit l'instruction IF X=1 THEN P:=0 ELSE P:=2;

Elle peut être représentée par le graphe de la figure n° 7.

Les deux arcs parallèles correspondant aux deux alternatives du IF sont d2 et d3

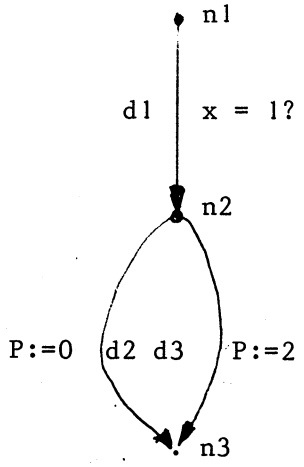


FIGURE N° 7

. Comme pour tous les graphes de calcul séquentiel, il n'y a qu'un seul noeud source et qu'un seul noeud puits.

Nous supposerons que pour chaque noeud il existe un chemin de la source au puits qui y passe : il n'y a ni partie inaccessible, ni circuit dont on ne peut sortir. Toutes les exécutions possibles se terminent normalement par le puits.

. Le problème de la reconnaissance des branches à partir du code d'un langage particulier peut, nous le verrons (chapitre IV), être facilement résolu à l'aide d'une analyse syntaxique.

A chaque type d'instruction, nous pouvons associer un sous-graphe particulier.

Le graphe peut alors être construit par imbrication de ces sous-graphes.

. Dans le cadre de la simulation qui nous intéresse, il existe un certain nombre de points où doit être réalisée la synchronisation entre les différents processus. Nous les figurerons au moyen de noeuds spéciaux notés ω.

. Nous pouvons associer à chaque arc dp, deux valeurs réelles :

- tp son temps d'exécution,
- Pp, la probabilité qu'il a d'être choisi après le passage par son noeud d'origine.

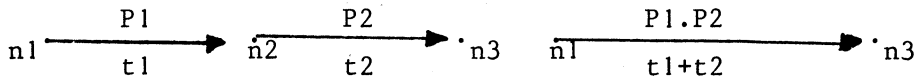
A l'aide de ces données, on peut faire le calcul du temps moyen d'exécution du programme représenté par le graphe (voir par exemple GRAHAM [G1])

Pour cela, nous faisons subir des réductions successives au graphe, en supprimant un à un les noeuds et en calculant à chaque pas les valeurs associées aux arcs remplaçant ceux qui ont été supprimés.

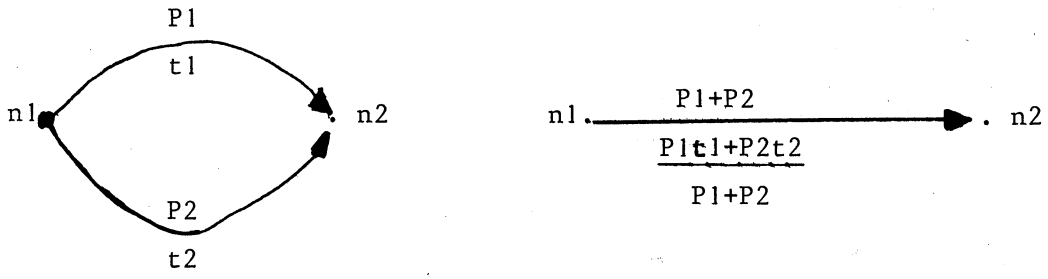
Lorsqu'il ne reste que la source et le puits, l'arc les reliant a comme valeurs : la probabilité 1, et le temps moyen d'exécution du graphe qu'il remplace.

Les réductions successives sont de trois types.

a) ARCS EN SERIE

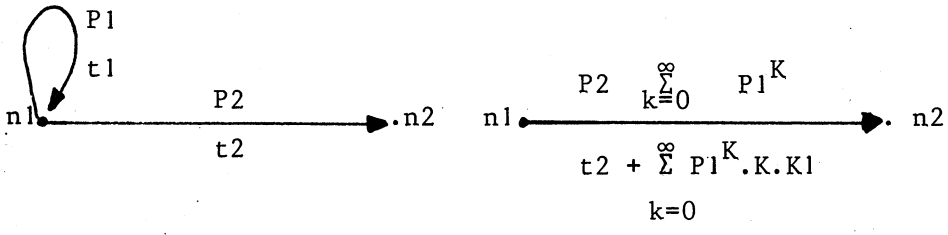


b) ARCS PARALLELES



c) SELF-BOUCLE

Une self-boucle est un arc ayant le même noeud pour origine et pour extrémité.



III - SOLUTIONS THEORIQUES DU PROBLEME

III.1. Notion de renseignement [K4]

Rappels d'algèbre :

Un dioïde est un ensemble muni de deux lois de composition interne :

La première notée + est associative et commutative

La seconde notée . est associative et distributive par rapport à la première.

Un élément blanc dans un dioïde est un élément noté B tel que pour tout a appartenant au dioïde :

$$\begin{cases} a + B = B + a = a \\ a \,.\, B = B \,.\, a = B \end{cases}$$

$$a + B = b + B \Rightarrow a = b$$

L'anneau des réels $\mathbb{R}$ constitue pour l'addition et la multiplication un dioïde qui possède un élément blanc : 0. Nous noterons $\mathbb{R}_B$ le dioïde formé par $\mathbb{R}$ auquel un élément blanc B a été ajouté : $\mathbb{R}_B = \mathbb{R} \cup B$.

C'est également un dioïde pour l'addition et la multiplication des réels.

Soit S_R l'ensemble des suites de réels défini ainsi :

. tout élément de S_R s'écrit (x1,x2,...,xn) avec xi $\in \mathbb{R}$ pour i = 1 à n

. L'ordre des réels n'intervient pas (x1,x2,x3), par exemple, représente le même élément de S_R que (x2,x1,x3)

. La suite vide notée ( ) appartient à S_R

L'ensemble S_R est un dioïde pour les lois de composition suivantes :

. La loi + définie ainsi :

$(x1,x2,\ldots,xn) + (y1,y2,\ldots,yp) = (x1,x2,\ldots,xn,y1,\ldots,yp)$

Nous l'appellerons concaténation

Il est aisé de voir qu'elle est commutative et associative.

. La loi . définie ainsi :

$(x1,\ldots,xn) \,.\, (y1,\ldots,yp) = (x1+y1,\ x2+y1,\ \ldots\ xn+y1,\ x1+y2,$
$x2+y2,\ \ldots\ x1+yp,\ x2+yp,\ \ldots\ xn+yp).$

+ désigne ici l'addition des réels. $(x1,\ldots,xn) \,.\, (\,) = (\,)$

Nous appellerons cette loi composition

Elle est commutative et distributive par rapport à la concaténation.

Le dioïde S_R possède un élément blanc : la suite vide.

DEFINITION : [K4]

Soit R1 une relation binaire définie sur un ensemble E. Une fonction F sera dite renseignement attachée à R1 si elle a les propriétés suivantes :

. Elle porte sur les mêmes variables que la relation R1 (c'est une fonction à deux variables)
. Les valeurs prises par F appartiennent à un dioïde D qui possède un élément blanc.
. Si un couple de variables ne vérifie pas la relation R1, la fonction F lui associe l'élément blanc et réciproquement.

Le dioïde D, qui est le domaine d'application de F, est appelé algèbre de renseignements

Soit un graphe $G = (X,\mathcal{U})$

Soit 0 la relation d'orientation définie ainsi sur X : Deux noeuds de X vérifient la relation 0 s'il existe au moins un arc allant du premier vers le second.

n1 0 n2 $\Longleftrightarrow \exists d \in \mathcal{U}$ tel que n1 soit son origine et n2 son extrémité.

REMARQUE

La donnée de la relation O, nous rapproche de la définition de BERGE [B1] qui est la suivante :

> Un graphe est la donnée d'un ensemble X et d'une relation Γ sur X.
>
> On le note $G = (X,\Gamma)$
>
> Les éléments de X sont appelés noeuds. Si ni et nj sont deux noeuds et si la relation ni Γ nj est vérifiée alors le doublet (ni,nj) constitue un arc allant de ni à nj.

La donnée de la relation Γ permet de définir au plus un arc par couple de noeuds. Aussi une telle définition n'est pas suffisante pour décrire les graphes de programme qui permettent l'existence d'arcs parallèles.

Les graphes de programmes sont en fait des multigraphes [H1] Ils peuvent être représentés par le quadruplet suivant :

$$G = (X, \mathcal{U}, f, s)$$

X : ensemble des noeuds

$\mathcal{U}$: ensemble des arcs

f : application de $\mathcal{U}$ dans X qui pour un arc donne son noeud d'origine

s : application de $\mathcal{U}$ dans X qui pour un arc donne son noeud d'extrémité.

Nous appellerons renseignement sur G, tout renseignement attaché à O.

EXEMPLE :

Nous noterons Rt le renseignement sur G à valeur dans le dioïde SR, qui à chaque couple de noeuds fait correspondre la suite éventuellement vide des temps d'exécutions associés aux différents arcs allant du premier noeud vers le second.

Nous pouvons représenter ce renseignement sous forme matricielle :

Les lignes et les colonnes représentent tous les noeuds du graphe. A l'intersection d'une ligne et d'une colonne, nous plaçons la valeur prise par le renseignement pour le couple de noeuds correspondant à cette ligne et cette colonne.

Le renseignement Rt, pour le graphe de la figure n° 8 , peut, par exemple être représenté par la matrice suivante :

	n1	n2	n3	n4	n5	n6
n1	()	(t1)	(t2)	()	()	()
n2	()	()	()	(t3)	()	()
n3	()	()	()	(t4)	(t5)	()
n4	()	(t6)	()	()	()	(t7)
n5	()	()	()	()	()	(t8,t9)
n6	()	()	()	()	()	()

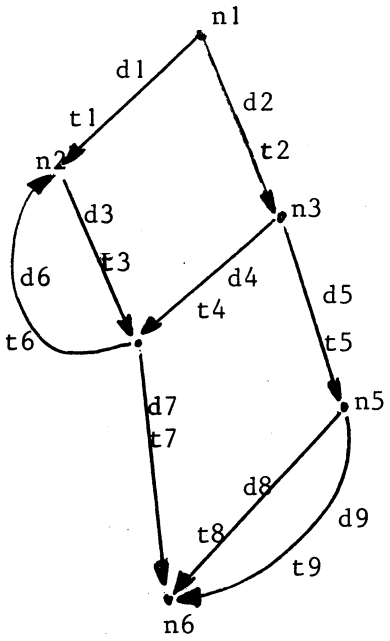


FIGURE N° 8

Les puissances successives de cette matrice dans SR donnent les temps d'exécution des chemins formés de 2 arcs puis 3, 4, ... etc.
En élevant au carré, on obtient, par exemple les temps d'exécution des chemins constitués de 2 arcs :

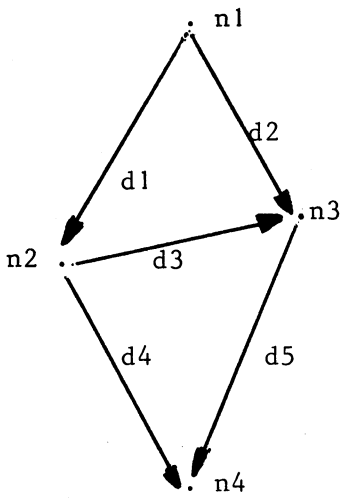
	n1	n2	n3	n4	n5	n6
n1	()	()	()	(t1+t3 t2+t4)	(t2+t5)	()
n2	()	(t3+t6)	()	()	()	(t3+t7)
n3	()	(t4+t6)	()	()	()	(t4+t7, t5+t8, t6+t9)
n4	()	()	()	(t3+t6)	()	()
n5	()	()	()	()	()	()
n6	()	()	()	()	()	()

REMARQUE :

Il serait intéressant de calculer la matrice $M^* = \sum_{k=1}^{\infty} M^k + I$ dont les éléments représentent les temps d'exécution de tous les chemins du graphe. Celui-ci possèdant des circuits, les éléments de M^* ne sont pas tous des suite finies. Nous pouvons cependant représenter toute suite de la forme (t1 + ... + tk), (t1+...+tk) + ... , n(t1+...+tk) , ...) par $(t1+...tk)^*$ Il existe plusieurs méthode pour construire la matrice M^*. Nous pouvons, par exemple, chercher à résoudre l'équation X = MX + R [V1]. Certains algorithmes procèdent par élimination successive des noeuds et se rapproche en celà du calcul du temps moyen d'exécution présenté par GRAHAM [G1].

III.2. Rappels de quelques notions de théorie des graphes

Les notations, que nous utiliserons, sont celles de BERGE [B1].



Soit G le graphe de la figure n° 4 :

$G = (X, \mathcal{U})$

avec $X = \{n1, n2, n3, n4\}$

$\mathcal{U} = \{d1, d2, d3, d4, d5\}$

Nous appellerons source un noeud n'ayant pas de prédécesseur et puits un noeud n'ayant pas de successeur. Etant donné un arc d allant de ni à nj, nous disons que ni est l'origine de d et nj son extrémité.

FIGURE N° 9

Le degré d'un noeud est le nombre d'arcs dont il est soit origine, soit extrémité.

Un chemin est une suite d'arcs d1 d2 ... dk telle que l'extrémité de l'un est l'origine du suivant.

Par exemple, dans le graphe de la figure n° 9, la suite d1d3d5 constitue un chemin qui va du noeud n1 au noeud n4.

Deux arcs sont dits adjacents, s'il existe un noeud qui soit à la fois origine ou extrémité pour chacun d'eux.

Par exemple, dans le graphe de la figure n° 9, d1 est adjacent à d2d3 et d4.

Une chaîne est une suite d'arcs d1d2 ...dk, telle que l'arc di relie le noeud ni au noeud ni+1 dans un sens ou dans l'autre.

La suite d2d3d4 constitue, dans la figure n° 9, une chaîne.

Un circuit est un chemin qui va d'un noeud sur lui-même. Un cycle est une chaîne dont le dernier arc est adjacent au premier.

La chaîne d1d3d2 forme un cycle.

Un graphe est dit connexe si pour tout couple de noeuds, il existe au moins une chaîne les reliant.

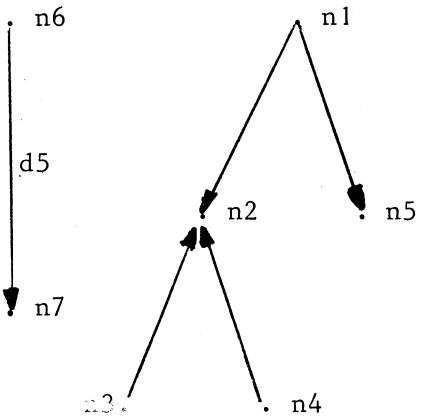


FIGURE N° 10

Le graphe de la figure n° 9 est connexe. Celui de la figure n°10 ne l'est pas.

Un arbre est un graphe connexe qui ne contient pas de cycle. La figure n° 10 est constituée de deux arbres.

Les trois propositions suivantes sont équivalentes :

a) G est un arbre
b) pour tout couple de noeuds, il existe une et une seule chaîne les reliant
c) la suppression d'un arc rompt la connexité.

Nous appellerons feuilles, les noeuds d'un arbre qui sont de degré 1.

Dans la figure n° 10, par exemple, les noeuds n3n4n5 et n7 sont des feuilles.

Soit G = $(X, \mathcal{U})$ un graphe.

Nous appellerons sous-graphe de G, tout graphe G' = $(X', \mathcal{U}')$ tel que

$X' \subseteq X$

$\mathcal{U}' \subseteq \mathcal{U}$

si d est un arc de $\mathcal{U}'$, ni son origine et nj son extrémité dans G appartiennent à X'.

Par exemple, dans la figure n° 10, les noeuds n1n2n5 constituent avec les arcs d1 et d2 un sous-graphe du graphe total.

Etant donné un graphe G connexe, nous appellerons arbre maximal sur G, tout sous-graphe de G qui possède tous les noeuds de G(X'=X) et qui soit un arbre.

Par exemple, dans la figure n° 9, les arcs d1 d3 et d4 constituent avec les noeuds qui leurs sont adjacents un arbre maximal.

III.3. Notion de répartition

Soit un graphe G = (X,$\mathcal{U}$)

Un métrage est une fonction M qui associe à chaque arc de G un nombre réel positif ou nul [K4].

Soit C un chemin de G.

C = d1 d2 d3 ... dp

Nous appellerons longueur du chemin C pour le métrage M, la valeur réelle X telle que : [K4]

X = M(d1) + M(d2) + ... + M(dp)

REMARQUE :

A tout métrage M, nous pouvons associer un renseignement R à valeur dans SR :

R fait correspondre, à chaque couple de noeuds, la suite éventuellement vide des valeurs prises par M pour les arcs allant du premier vers le second.

M peut alors être représenté par la matrice qui décrit le renseignement R.

Les puissances de cette matrice donnent les longueurs des différents chemins de G, pour le métrage M.

Par exemple Mt, le métrage qui fournit pour chaque arc son temps d'exécution, correspond au renseignement Rt qui a été décrit au paragraphe précédent.

DEFINITION

Nous appellerons répartition sur G, tout métrage pour lequel les chemins allant de la source au puits ou à un noeud ω ont même longueur que pour Mt, le métrage qui associe à chaque arc son temps d'exécution.

REMARQUES

-- La longueur d'un chemin pour Mt n'est autre que son temps d'exécution.

Etant donnés un graphe de programme G et une répartition F sur G, si en parcourant le graphe à partir de la source, nous faisons la somme des valeurs non nulles de F pour les différents arcs utilisés, au passage par les noeuds ω et à l'arrivée au puits, nous obtenons le temps réel où en est l'exécution.

Le problème de minimiser les appels de remise à jour est donc équivalent à celui de trouver une répartition pour laquelle un maximum d'arcs ont une valeur nulle.

Une telle répartition sera dite minimale en nombre.

-- Le métrage Mt constitue lui-même une répartition. Nous l'appellerons répartition réelle.

EXEMPLE DE REPARTITION :

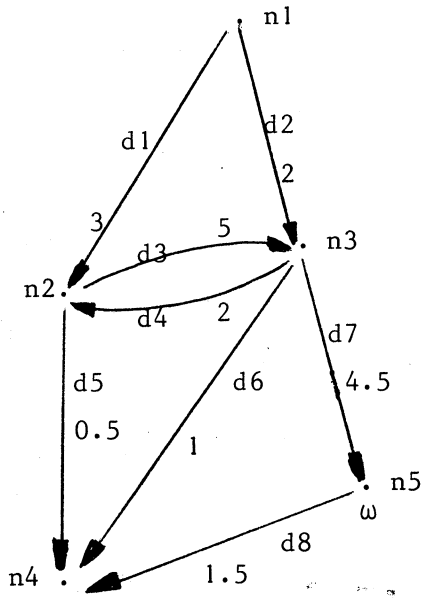


FIGURE N° 11

Soit le graphe de la figure n° 11, nous avons inscrit sur chaque arc son temps d'exécution. Les chemins pour lesquels la répartition doit fournir la même longueur que Mt sont de la forme :

$$\left[\begin{array}{l} d1\ (d3d4)^n\ d5 \\ d1\ (d3d4)^n\ d3d6 \\ d1\ (d3d4)^n\ d3d7d8 \\ d1\ (d3d4)^n\ d3d7 \\ d2\ (d4d3)^n\ d7 \\ d2\ (d4d3)^n\ d7d8 \\ d2\ (d4d3)^n\ d6 \\ d2\ (d4d3)^n\ d4d5 \end{array}\right.$$

$(d3d4)^n$ indique le chemin

$$\underbrace{d3d4\ d3d4\ \ldots\ d3d4}_{n\ \text{fois}}$$

Soit le métrage suivant :

M(d1) = 0	M(d5) = 3.5
M(d2) = 2	M(d6) = 1
M(d3) = 8	M(d7) = 4.5
M(d4) = - 1	M(d8) = 1.5

C'est une répartition.

Soit, par exemple c = d1 d3 d4 d3 d4 d3 d7 un chemin parmi ceux précédemment cités.

La longueur de C pour M est :

V = M(d1) + 3 M(d3) + 2 M(d4) + M(d7)

= 0 + 3 X 8 + 2 X(-1) + 4.5 = $\boxed{26.5}$

La longueur de C pour Mt est :

Vt = 3 + 3 X 5 + 2 X 2 + 4.5 = $\boxed{26.5}$

Elles sont bien égales.

Etant donné un graphe de programme G = (X, $\mathcal{U}$) nous noterons:

W' , le sous ensemble de X formé par la source, le puits et les noeuds ω

$\mathcal{F}G$, l'ensemble des répartitions sur G

MG , l'ensemble des métrages sur G

TV, ni, l'application de $\mathcal{F}G$ dans MG définie ainsi :

$$F \in \mathcal{F}G \xrightarrow{TV,\ ni} M \in MG \quad V \in \mathcal{R} \text{ et } ni \in X-W'$$

M(dK) = F(dK)	pour tout arc dK non adjacent à ni ou ayant ni comme origine et extrémité
M(dK) = F(dK) - V	pour tout arc dK ayant ni comme extrémité, et non comme origine
M(dK) = F(dK) + V	pour tout arc dK ayant ni comme origine, et non comme extrémité.

L'application TV,ni modifie les valeurs associées aux arcs adjacents à ni en retranchant V pour ceux qui arrivent en ajoutant V pour ceux qui en sortent.

T , l'ensemble des applications TV,ni avec $v \in \mathbb{R}$, $v \geq 0$, $ni \in X-W'$
TO,ni, sera quelque soit le noeud ni de X-W', l'application identité.

A une **répartition** F, elle fait correspondre elle-même :
$TO,ni(F) = F \quad \forall ni \in X-W'$.

THEOREME I

Le domaine d'application de TV,ni appartenant à T est contenu dans $\mathcal{F}G$:

$$\forall v \in \mathbb{R}, \forall ni \in X-W', \forall F \in \mathcal{F}G$$
$$TV,ni(F) \in \mathcal{F}G$$

L'application TV,ni définit une transformation élémentaire pour passer d'une répartition à une autre.

DEMONSTRATION :

Soit F une répartition sur G : M = TV,ni(F).

Soit C = d1 d2 ...dn un chemin de la source à un noeud de W'. Deux cas peuvent se produire :

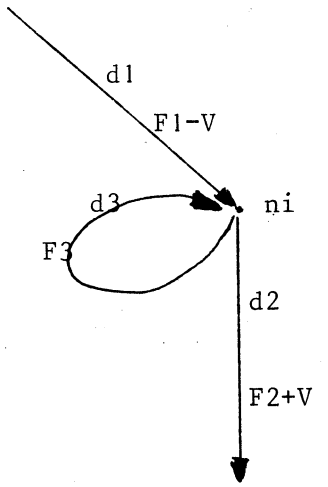


FIGURE N° 12

1) C ne passe pas par le noeud ni

La longueur est alors la même pour F et M : car pour tout arc d de C, M(d) = F(d)

2) C passe par ni

ni n'appartient pas à W' donc chaque fois que l'on y entre on en sort. Les arcs adjacents à ni sont soit des self boucles, soit associables par couple dont l'un arrive et l'autre part. Soit d1 d2 un tel couple

$M(d1) + M(d2) = F(d2)-V + F(d2)+V$
$= F(d1) + F(d2)$

Les autres arcs se voyant associés la même valeur par M et F. La longueur de C est la même pour ces deux métrages.

En définitif, pour tout chemin de la source au puits ou à un noeud ω, la longueur est la même pour les métrages M et F. M est une répartition.

LEMME 1

Soit C un chemin reliant deux noeuds de W'. La longueur de C est la même pour toutes les répartitions. Elle peut en effet être considérée comme la différence des longueurs de deux chemins allant de la source à un noeud de ω'.

THEOREME II

Soient F1 et F2 deux répartitions sur G, F2 peut être obtenue à partir de F1 par un nombre fini d'applications de T: il suffit d'une application, qui peut être éventuellement l'identité, par noeud de X - W'.

DEMONSTRATION :

Numérotons les noeuds de X-W' de façon que chacun d'eux ait au moins un prédécesseur soit de numéro inférieur, soit contenu dans W'.

Celà peut être aisément obtenu, en balayant le graphe à partir de la source et en numérotant les noeuds de X- ω' au fur et à mesure de leur rencontre.

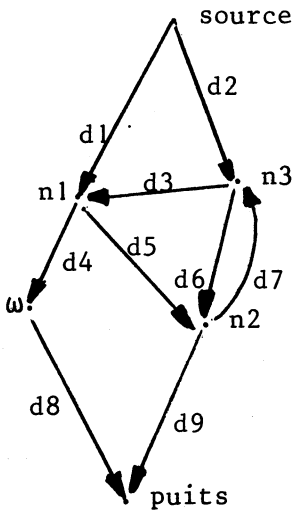


FIGURE N° 13

Pour la figure n° 13, par exemple, nous avons parcouru les chemins : d1 d4 d8, d1 d5 d9, et d2. Soient F1 et F2 deux répartitions sur G.

Soit n1 n2... nk les noeuds de X-W' numérotés comme précédemment.

Nous appliquons à F1 la suite d'éléments de T, TV1, n1, TV2, n2 ... TVk, nk, définie ainsi :

Etant donné ni, la valeur vi sera donnée par l'égalité :

vi = TVi,ni-1 (TVi-2 ni-2 (...
(TV1,n1(F1))...))(d) - F2(d)

avec d l'arc qui arrive à ni d'un noeud de W' ou bien d'un noeud de numéro inférieur à i.

L'application TVi,ni rend la valeur associée à l'arc d égale à ce qu'elle est pour la répartition F2.

La nouvelle valeur associée à d ne sera pas modifiée par les applications suivantes : si n, l'origine de d, est un noeud de X-W', l'application TV,n a déjà été utilisée.

Soit F' la répartition ainsi construite

Soit A l'ensemble des arcs ayant mêmes valeurs pour F' et F2.

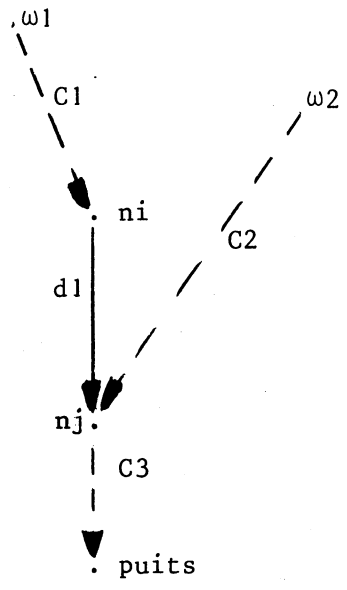
Appelons chemin vide, un chemin constitué par aucun arc. Etant donné un chemin C, nous noterons LF'(C) la longueur de C pour F' et LF2(C) celle pour F2. Soit un arc quelconque d1. Notons ni son origine, et nj son extrémité.

Nous pouvons construire à partir de ni, en prenant successivement des arcs de A (arcs provenant d'un noeud de numéro inférieur dont la valeur pour F' a été rendue égale à celle pour F2), un chemin, éventuellement vide C1 allant d'un noeud de W' à ni.

Soit C2 le chemin construit comme précédemment à partir de nj.

Soit C3 un chemin de nj au puits.

Nous avons :



$F'(d1) = LF'(C1\ d1\ C3) - LF'(C1) - LF'(C3)$

or $LF'(C3) = LF'(C2\ C3) - LF'(C2)$

donc $F'(d1) = LF'(C1\ d1\ C3) - LF'(C1) - LF'(C2\ C3) + LF'(C2)$

Les chemins C1 d1 C3 et C2 C3 relient deux noeuds de W'; les chemins C1 et C2 sont formés par des arcs de A. Ils ont tous même longueur pour F' et F2 donc

$F'(d1) = LF2(C1\ d1\ C3) - LF2(C1) - LF2(C2\ C3) + LF2(C2) = F2(d1)$

F' n'est autre que la répartition F2.

FIGURE N° 14

III.4. Notion de graphe réduit

Etant donné un graphe de programme G = (X,$\mathcal{U}$), nous appellerons graphe réduit de G le graphe G' = (X', $\mathcal{U}'$) défini ainsi :

X' = (X - W') $\cup$ {W1} (W1 est un noeud)

$\mathcal{U}'$ = ($\mathcal{U}$ - E) $\cup$ E'

avec E l'ensemble des arcs adjacents à un noeud de W'.

E' est obtenu à partir de E en remplaçant pour chaque arc les noeuds de W' par W1.

G' est en fait obtenu à partir de G en confondant tous les noeuds de W' en un seul et même noeud que nous noterons W1. Cette transformation sera appelée réduction.

EXEMPLE :

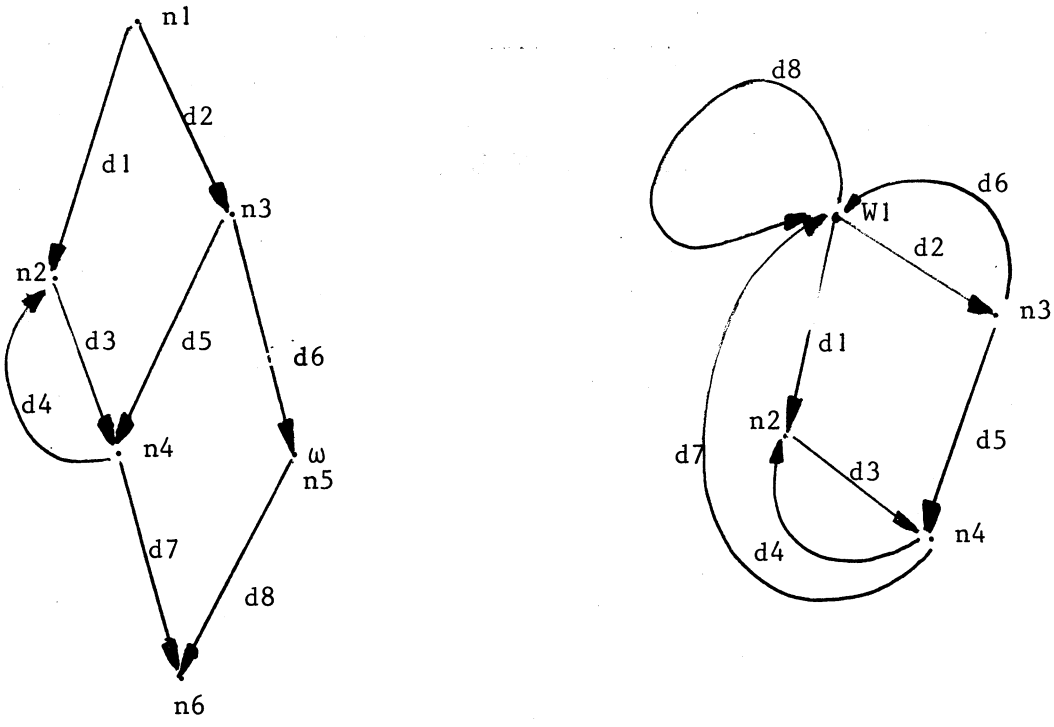


FIGURE N° 14

REMARQUES

-- Tous les arcs de G se retrouvent, à une transformation près, dans le graphe réduit G'. Cette transformation consiste à remplacer par W1 les origines ou extrémités des arcs qui sont des noeuds de W'. Nous considérerons que ce sont les mêmes arcs.

-- Le graphe réduit G' ne possède ni source ni puits. Les chemins de la source au puits dans le graphe G, qui représentent en fait les exécutions possibles, correspondent dans G' à des circuits partant de W1.

Cependant tous les chemins de G' partant de W1 et y aboutissant ne forment pas forcément dans G une exécution possible du graphe. Ils peuvent représenter un chemin reliant deux noeuds de W'.

Nous appellerons répartitions sur un graphe réduit G', tout métrage sur G' tel que les longueurs de tous les circuits partant de W1 sont les mêmes que pour Mt, le métrage qui associe à chaque arc son temps d'exécution dans G.

THEOREME III

Toute répartition sur G fournit également une répartition sur G'.

Et réciproquement, toute répartition sur G' fournit une répartition sur G.

DEMONSTRATION

Soit F' une répartition sur G'.

Tous les chemins de la source à un noeud de W', dans G, correspondent à un circuit sur W1 dans G'.

Leurs longueurs sont les mêmes pour F' et Mt. F' est une répartition sur G.

Soit F une répartition sur G. D'après le lemme 1, tous les chemins reliant deux noeuds de W' ont même longueur pour F et Mt.

Dans G' tous les chemins de W1 à W1 ont donc même longueur pour F et Mt.

F est une répartition sur G'.

Dorénavant, nous utiliserons de préférence le graphe réduit pour représenter un programme. Il a l'avantage de n'avoir qu'un seul noeud pour figurer l'ensemble W'. Celà facilite l'application d'algorithme qui, comme celui décrit au chapitre précédent, mettent à part les noeuds de W'.

III.5. Notions d'ensemble d'arcs supprimable

III.5.1. Définition et propriétés

Soit un graphe de programme G = (X, $\mathcal{U}$).

Soit D un sous-ensemble de $\mathcal{U}$.

Notons t1 t2 ... tn les temps d'exécutions des arcs de $\mathcal{U}$.

D sera dit supprimable si quelque soient les valeurs de t1 t2 ... tn, il existe une répartition F sur G telle que pour tout arc d de D, on ait :

$$F(d) = 0$$

Pour un même ensemble d'arcs supprimable D, il peut évidemment exister plusieurs répartitions ayant cette propriété.

Par exemple, pour le graphe de la figure n° 15 :
Soit l'ensemble D = {d2} si t1 t2 t3 et t4 sont les temps d'exécutions de d1 d2 d3 et d4, les répartitions F1 et F2 décrites ci-dessous associent toutes deux la valeur 0 à l'arc d2.

F1 : F1(d1) = t1 + t2 F1(d3) = t3-t2
F1(d2) = 0 F1(d4) = t4
F2 : F2(d1) = t1 F2(d3) = t3-t2
F2(d2) = 0 F2(d4) = t4+t2

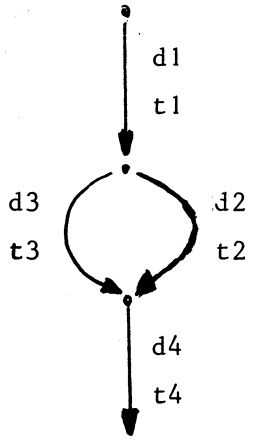


FIGURE N° 15

REMARQUE

Les valeurs des temps d'exécutions des arcs de G, t1, t2 ..., tn, dépendent de la machine simulée, aussi nous les supposerons inconnus au moment du choix des endroits où seront implantés les appels. Dans ce cas, le problème de minimiser le nombre de ces appels est équivalent à celui de trouver un ensemble d'arcs supprimable maximum en taille.

Notons G'D le sous-graphe de G' formé par les arcs de D.

THEOREME IV

Soit D un ensemble d'arcs.

Pour que D soit supprimable, il faut et il suffit qu'aucun sous-ensemble de D ne constitue, dans G', un cycle. G'D est alors formé par un ou plusieurs arbres.

DEMONSTRATION

CN : Soit D un ensemble supprimable d'arcs de G'.

Supposons que D contienne un cycle CD de G'.

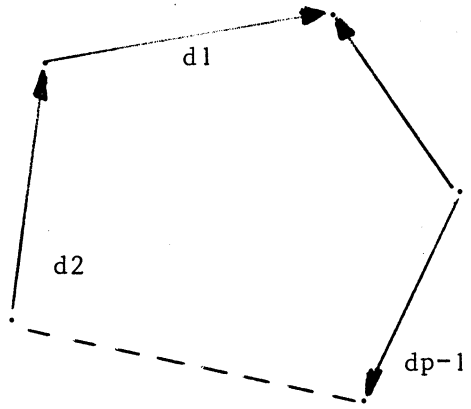
CD = d1 d2 ... dp

D'après le théorème II nous pouvons en partant de la répartition réelle, annuler les valeurs associées aux arcs de CD en faisant une transformation par noeud de CD, excepté W1. Si W1 appartient à CD, ce n'est pas possible car il faudrait annuler alors p valeurs distinctes par p-1 applications.

Dans le cas contraire, numérotons les noeuds de CD de la manière suivante :

} ni est le noeud adjacent à di et di+1
} np est le noeud adjacent à dp et d1

La figure n° 15 représente le cycle CD avec cette numérotation des noeuds.



Soient TV1,n1 ... TVp,np les applications de T qui rendent nulles les valeurs associées aux arcs d1 d2...dp
Notons (t1,t2,...,tp) les valeurs associées par Mt aux arcs de CD.
Nous allons voir les transformations apportées à cette suite, par
TV1,n1 TV2,n2 ... TVp,np.

FIGURE N° 15

$(t1,t2,\ldots,tp) \xrightarrow{TV1,n1} (0,t1 \pm t1,t,\ldots,tp) \longmapsto$

$\xrightarrow{TV2,n2} (0,0,t3 \pm t2 \pm t1,\ t4,\ldots,tp)$

.....

$\longmapsto \xrightarrow{TVp-1,np-1} (0,0,\ldots,0,\ tp \pm tp-1 \pm \ldots \pm t1)$

La transformation TVp,np ne peut pas rendre nulle la valeur associée à dp sans remettre celle associée à d1 à une valeur non nulle :

$\pm$ tp $\pm$ tp-1 ... $\pm$ t1

C'est donc impossible.

CS : Soit D un ensemble d'arcs de G' ne contenant pas de cycle.
G'D est formé par un ou plusieurs arbres.

Le noeud W1 appartient au plus à l'un d'entre eux. Nous allons voir que pour chacun d'eux, nous pouvons annuler les valeurs de toutes les branches par l'itération I1 décrite plus loin.

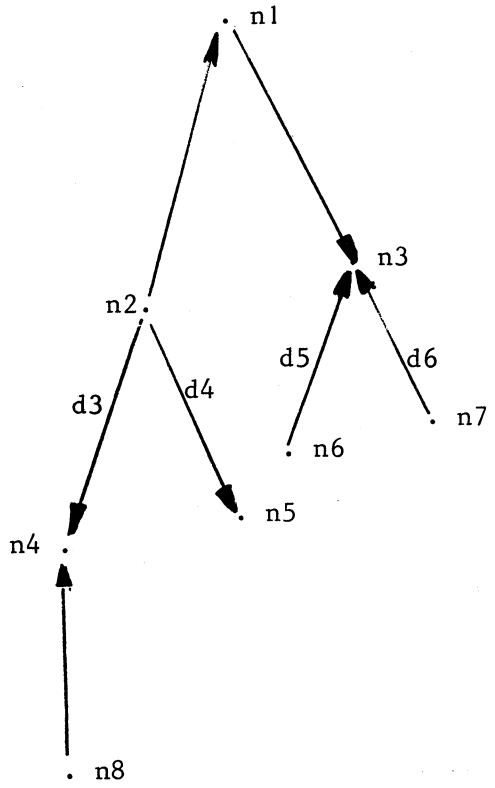


FIGURE N° 16

Soit A un arbre de G'D.
Si W1 appartient à A, nous le noterons n1, sinon n1 sera n'importe quel noeud de A. Nous appellerons niveau d'un noeud par rapport à n1 le nombre d'arcs que possède la chaîne le reliant à n1 [K4]
Numérotons n1 n2 n3 ... nk, les noeuds de A par ordre croissant de niveau par rapport à n1.
Le graphe de la figure n° 16 montre une telle numérotation des noeuds.

L'itération I1 consiste alors à appliquer TV2, n2 TV3, n3 ... TVk,nk, les éléments de T qui annulent les valeurs des arcs provenant d'un noeud de niveau inférieur. Les valeurs ainsi annulées ne seront plus modifiées car un noeud de niveau inférieur porte forcément un numéro inférieur.

En appliquant I1 aux différents arbres de G'D nous obtenons une répartition pour laquelle les arcs de D se voient associés des valeurs nulles et celà quelques soient les temps d'exécution des arcs.
D est supprimable.

CORROLLAIRE

Tout ensemble supprimable maximum en taille D est tel que G'D constitue un arbre maximal de G'. Réciproquement, tout arbre maximal de G' est tel que l'ensemble de ses arcs forme un ensemble supprimable maximum en taille.

III.5.2. Problème de KNUTH

Le corollaire du théorème IV a quelques similitudes avec les résultats présentés par KNUTH à propos d'un problème assez voisin du notre [K2].

Il cherche à déterminer le nombre d'utilisations de chaque branche d'un graphe de programme, au cours d'une exécution, en appliquant la loi de KIRCHOFF qui peut s'énoncer ainsi :

Chaque fois que nous entrons dans un **noeud**,nous en sortons.

Pour ce faire, la loi de KIRCHOFF ne permettant pas de résoudre toutes les inconnues du problème, il cherche à déterminer quel est le plus petit sous-ensemble des arcs du graphe dont les nombres de passages permettent de calculer tous les autres : c'est-à-dire l'ensemble des arcs qui permettent à eux seuls de distinguer toutes les exécutions possibles du graphe. Etant donné un arbre maximal de G, l'ensemble cherché n'est autre que celui formé par les arcs de G n'appartenant pas à l'arbre.

Nous allons étudier maintenant le théorème énoncé par KNUTH et voir comment nous pouvons l'appliquer à la résolution de notre problème.

III.5.2.1. Complèment de définition [K2]

Soit une chaîne C = d1 d2 d3 d4 d5. L'ordre dans lequel sont écrits les arcs indique un sens de parcours de la chaîne. Pour tenir compte de l'orientation de chacun de ces arcs nous noterons -d ceux parcours en sens inverse et +d les autres.

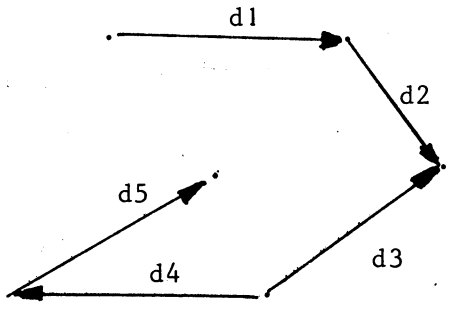


FIGURE N° 17

La chaîne C s'écrit alors : C = +d1 +d2 -d3 +d4 + d5.
Nous dirons que la chaîne C parcours 1 fois les arcs d1 d2 d4 d5 et -1 fois l'arc d3.

-- Soit M un métrage sur le graphe G.

Nous noterons LM(C), la longueur de la chaîne C par rapport à M, qui est définie ainsi :

$$LM(C) = \sum_{d \in E1} M(d) - \sum_{d \in E2} M(d)$$

avec E1 l'ensemble des arcs parcourus dans le bon sens par C et E2 l'ensemble de ceux qui le sont en sens inverse.

-- Soit $\mathcal{C}$ un ensemble de chaîne de G.

Le nombre de passages d'un arc d de G pour $\mathcal{C}$ sera la valeur de la somme des nombres de passages éventuellement négatifs, des différentes chaînes de $\mathcal{C}$ par l'arc d.

Soit, par exemple, $\mathcal{C}$ = {+d1 -d2, +d2 -d3}
le nombre de passage par d2 pour $\mathcal{C}$ est égale à -1+1 soit 0.

-- Soit Γ un chemin de G.

Soit $\mathcal{C}$ un ensemble de chaînes de G.

Nous dirons que $\mathcal{C}$ est équivalent à Γ, pour les nombres de passages par les arcs, si pour tout arc d de G le nombre d'occurences de d dans Γ est égal au nombre de passage de d pour $\mathcal{C}$.

Par exemple, Γ = d1 d2 d3 est équivalent à

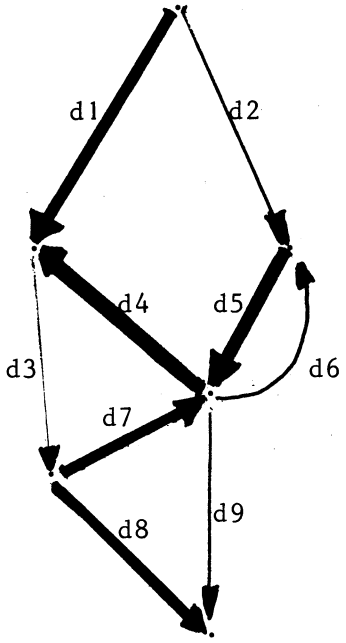
$\mathcal{C}$ = {d1 - d4 + d3, +d2 + d4}

REMARQUE :

Soit un métrage M sur G.

Soit un chemin Γ et un ensemble de chaînes $\mathcal{C}$ équivalent à Γ pour les nombres de passages par les arcs. La longueur de Γ pour M est égale à la somme des longueurs des chaînes de $\mathcal{C}$.

-- Soit G un graphe et A un arbre maximal sur G.
Soit d un arc quelconque n'appartenant pas à A. Nous appellerons cycle fondamental le cycle unique que constitue d avec les arcs de A [K2].



Les arcs de A sont en trait gras

FIGURE N° 19

Nous le noterons en respectant le sens de parcours de d et en commençant par cet arc. Si G possède n noeuds et m arcs, nous pouvons trouver m-n+1 cycles fondamentaux : c'est-à-dire autant que d'arcs de G n'appartenant pas à A.
Nous appellerons chaîne fondamentale la chaîne de A qui relie **la source** au puits.
Par exemple, pour le graphe de la figure n° 19 la chaîne fondamentale s'écrit:
+d1-d4 - d7+d8
Les cycles fondamentaux sont :

+d2+d5+d4-d1, pour l'arc d2
+d3+d7+d4, pour d3
+d6+d5, pour d6
+d9-d8+d7, pour d9

III.5.2.2. Théorème

Soient G un graphe de programme et A un arbre maximal sur G. Tout chemin de la source au puits, est équivalent pour les nombres de passages par les arcs à un ensemble unique formé par la chaîne fondamentale et un nombre déterminé de fois chaque cycle fondamental.

Réciproquement, tout ensemble formé par la chaîne fondamentale et un certain nombre de fois chaque cycle fondamental, tel que les nombres de passages par les arcs soient des entiers positifs ou nuls, est équivalent à un chemin unique de la source au puits.

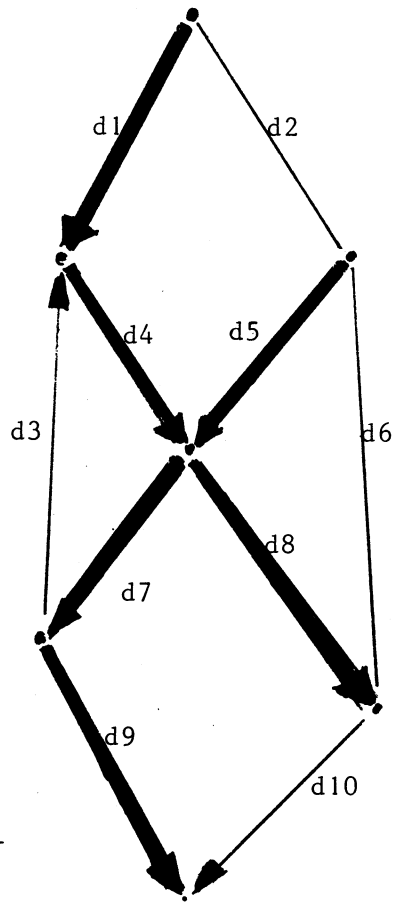
Nous allons voir, avec l'appui d'un exemple, comment à partir d'un chemin Γ de la source au puits, nous pouvons construire l'ensemble $\mathcal{C}$ qui lui est équivalent pour les nombres de passages par les arcs.

Soit d un arc n'appartenant pas à l'arbre maximal A. d ne figure pas dans la chaîne fondamentale, ni dans aucun autre cycle fondamental que celui qu'il constitue avec les arcs de A.

Aussi, pour chaque occurrence de d dans Γ, il y aura une occurrence du cycle foncamental qui lui correspond dans l'ensemble $\mathcal{C}$.

Il suffit donc, pour construire $\mathcal{C}$ de déterminer dans Γ toutes les occurrences des arcs n'appartenant pas à A.

Soit, par exemple, le graphe de la figure n° 20



en trait gras les arcs de A

FIGURE N° 20

La chaîne fondamentale est :
+d1+d4+d7+d9

Les cycles fondamentaux sont :
+d2+d5-d4-d1
+d3+d4+d7
+d6-d8-d5
+d10-d9-d7+d8

Soit le chemin :
Γ = d1 d4 d7 <u>d3</u> d4 d8 <u>d10</u>

Sont soulignés les occurrences des arcs n'appartenant pas à A. Par la méthode précédemment exposée, nous construisons l'ensemble $\mathcal{C}$ suivant :

$\mathcal{C}$ = {+d1+d4+d7+d9,
+d3+d4+d7,
+d10-d9-d7+d8}

Nous pouvons voir aisément que pour tout arc, le nombre de passages dans $\mathcal{C}$ est égal au nombre d'occurrences dans Γ.

Nous allons étendre ce théorème au cas des graphes réduits. Dans ce cas, il n'y a plus ni source, ni puits et les chemins qui nous intéressent, sont ceux allant de W1 à W1 (circuits sur W1).

La chaîne fondamentale devient alors un cycle fondamental, au même titre que les autres.

Le théorème devient :

Soit G' le graphe réduit d'un graphe de programme G.

Soit A un arbre maximal sur G'.

Tour circuit sur W1 est équivalent pour les nombres de passages par les arcs à un ensemble unique formé par un cycle fondamental passant par W1 plus un nombre déterminé de fois chaque cycle fondamental.

Réciproquement, tout ensemble formé par un cycle passant par W1 et par un nombre déterminé de fois chaque cycle fondamental, tel que les nombres de passages par les arcs soient des entiers positifs ou nul, est équivalent pour ces nombres à un circuit unique sur W1.

Soit Γ un circuit sur W1, nous pouvons utiliser la même méthode que dans le cas précédent, pour construire l'ensemble qui lui est équivalent. Sa chaîne fondamentale est alors seulement devenue un cycle fondamental semblable aux autres.

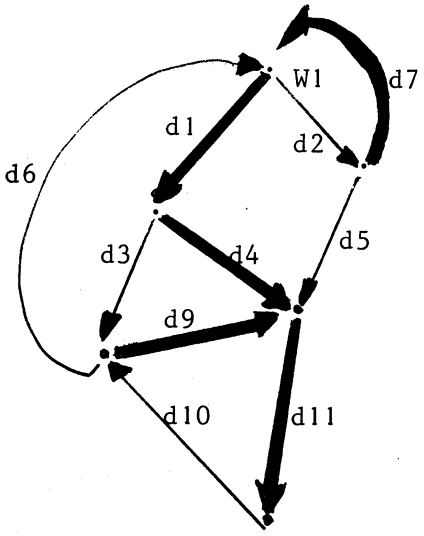


FIGURE N° 21

Soit, par exemple, le graphe réduit G' représenté par la figure n° 21. Les cycles fondamentaux sont :

$$\begin{bmatrix} +d2+d7 \\ +d3+d9-d4 \\ +d5-d4-d1-d7 \\ +d6+d1+d4-d9 \\ +d10+d9+d11 \end{bmatrix}$$

Soit Γ= d1<u>d3d6d2d5</u>d11<u>d10d6</u>

Nous avons souligné les arcs n'appartenant pas à A. A chacun d'eux, nous associons dans Γ le cycle fondamental qu'il forme avec les arcs de A.

$\mathcal{C}$ = {+d3+d9-d4, +d6+d1+d4-d9, +d2+d7, +d5-d4-d1-d7,
+d10+d9+d11, +d6+d1+d4-d9}

Nous pouvons aisément constater que $\mathcal{C}$ est bien équivalent à Γ.

REMARQUE :

Soit Mt le métrage sur G' qui associe à chaque arc son temps d'exécution.

Soit Γ un circuit sur W1 et $\mathcal{C}$ l'ensemble de cycles fondamentaux qui lui est équivalent.

La longueur de Γ pour Mt est égale à la somme des longueurs des cycles de $\mathcal{C}$.

Or, il y a une correspondance bijective entre les occurences des arcs n'appartenant pas à A dans Γ et les cycles de $\mathcal{C}$.

Nous pouvons déduire de tout celà, un algorithme I2, permettant d'obtenir, à partir d'un ensemble D d'arcs supprimables qui est maximum en taille, une répartition F qui associe la valeur 0 à tout arc de D.

I2 :

Soit G'D l'arbre maximal de G' constitué par les arcs de D.

Nous construisons F de la manière suivante :

$F(d) = 0 \quad \forall d \in D$

$F(d) = LMt(Cd)$ avec Cd le cycle fondamental formé par d et les arcs de G'I $\forall d \notin D$.

(LMt(Cd) est la longueur pour Mt du cycle Cd). F est bien une répartition :

La longueur de tout circuit sur W1, pour le métrage F, est égale à la somme des longueurs des cycles fondamentaux de $\mathcal{C}$, l'ensemble qui lui est équivalent, pour le métrage Mt. Donc la longueur de tout circuit sur W1 pour F est égale à sa longueur pour Mt.

III.5.2.3. Construction des cycles fondamentaux

Le seul problème qu'il reste à résoudre est celui de construire le cycle fondamental correspondant à un arc d n'appartenant pas à D.

Il s'agit en fait de trouver le cycle unique de G'Dud, le sous-graphe de G' constitués par les arcs de Dud. C'est un problème classique de la théorie des graphes. Nous allons brièvement exposer ici deux méthodes parmi celles existantes.

III.5.2.3.1. La méthode matricielle

Soit S l'ensemble des suites d'arcs de G'Dud.

Nous appellerons concaténation la loi de composition interne sur S qui à deux éléments $S1 = d_1^1\ d_2^1\ d_3^1\ d_n^1$ et $S2 = d_1^2\ d_1^2\ d_3^2 \ldots d_p^2$ fait correspondre l'éléments $s = s1 \,.\, s2 = d_1^1\ d_2^1 \ldots d_n^1\ d_1^2 \ldots d_p^2$ avec la propriété suivante : pour tout arc di de Dud, di.di = V la suite vide.

- soit D = {d1 d2 d3 d4}

  s1 = d1 d3 d5 d2

  s2 = d2 d5 d3 d4

  s1.s2 = d1 d3 d5 d2 d2 d5 d3 d4 = d1 d3 d5 d5 d3 d4

  = d1 d3 d3 d4 = **d1 d4**

Soit S' l'ensemble des sous-ensembles de S. Un élément de S' sera constitué par un certain nombre éventuellement nul, de suites de S.

S' est un dioïde pour les deux lois de composition interne suivante :

-- La loi notée + qui associe à deux éléments de S' le sous-ensemble de S formé par leur union

-- La loi notée X définie ainsi :

soient $s'1 = \{s_1^1, s_2^1, s_3^1, \ldots, s_n^1\}$ et

$$s'2 = \{s_1^2, s_2^2, s_3^2, \ldots, s_p^2\}.$$

deux éléments de S' :

$$s'_1 \,.\, s'_2 = \{s_1^1.s_1^2,\ s_2^1\ s_1^2, \ldots, s_n^1\ s_1^2$$

$$s_1^1\ s_2^2, \ldots,\ s_n^1\ s_2^2, \ldots,\ s_1^1\ s_p^2,\ \ldots,\ s_n^1\ s_p^2\}$$

avec $s_i^1.s_j^2$ la concaténation de s_i^1 et s_j^2.

Le dioïde possède un élément blanc qui est $\emptyset$ l'ensemble vide :

$$\begin{cases} s' + \emptyset = s' & \forall\, s' \in S \\ s' \times \emptyset = \emptyset & \forall\, s' \in S \end{cases}$$

Soit M, la matrice construite de la manière suivante : les lignes et les colonnes représentent respectivement tous les noeuds de G'Dud préalablement numérotés. A l'intersection de la $i^{ème}$ ligne et la $j^{ème}$ colonne, nous inscrivons l'élément mij définie ainsi :

$$\begin{cases} mij = \emptyset & \text{s'il n'existe pas d'arc reliant les noeuds numérotés i et j.} \\ mij = \{d1, d2, \ldots, dp\} & \text{sinon avec d1 d2 dp, les arcs reliant le noeud numéroté i au noeud numéroté j sans tenir compte de leur orientation.} \end{cases}$$

Les puissances de M dans le dioïde S' donnent les différentes chaînes du graphe entre le noeud de la ligne et celui de la colonne.

Pour avoir le cycle unique qui nous intéresse, il suffit d'élever M aux puissances successives jusqu'à ce qu'il apparaisse un élément sur la diagonale.

Cet élément est le cycle cherché.

EXEMPLE :

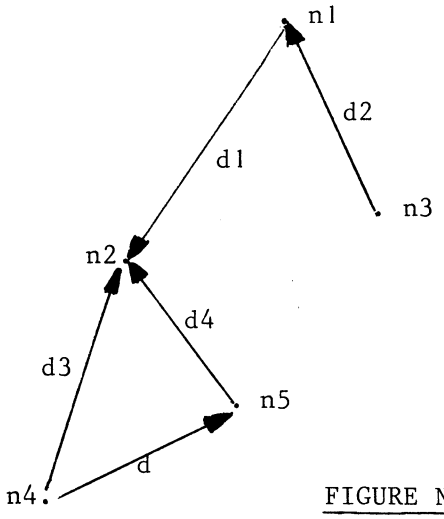


FIGURE N° 22

Soit le graphe de la figure n° 22.

La matrice M est la suivante :

	n1	n2	n3	n4	n5
n1	$\emptyset$	{d1}	{d2}	$\emptyset$	$\emptyset$
n2	{d1}	$\emptyset$	$\emptyset$	{d3}	{d4}
n3	{d2}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n4	$\emptyset$	{d3}	$\emptyset$	$\emptyset$	{d}
n5	$\emptyset$	{d4}	$\emptyset$	{d}	$\emptyset$

M^3 =

	n1	n2	n3	n4	n5
n1	∅	{d1}	{d2}	{d1d4d}	{d1d3d}
n2	{d1}	{d3dd4}	∅	{d3}	{d4}
n3	{d2}	∅	∅	{d2d1d3}	{d2d1d4}
n4	{dd4d1}	{d3}	{d3d1d2}	{d3d4d}	{d}
n5	{dd3d1}	{d4}	{d4d1d2}	{d}	{d4d3d}

Le cycle cherché s'écrit donc : +d+d4-d3

III.5.2.3.2. La méthode combinatoire

Elle consiste à balayer toutes les chaînes du graphe à partir de l'arc d.

Etant donné une chaîne Γ = d1 d2 ... dp, nous appellerons noeud libre de dp, l'origine ou l'extrémité de dp qui est adjacente à aucun autre arc de Γ.

Pour construire le cycle C, par cette méthode, nous pouvons appliquer l'algorithme de pile suivant :

Au départ C = d.

a) Nous lui ajoutons un arc d' adjacent au noeud libre de son dernier arc.

Au départ, l'origine et l'extrémité de d sont des noeuds libres. Nous choisirons son extrémité.

Si l'autre extrémité de d' est une feuille nous passons au pas b. Si d' est adjacent à l'origine de d, C est le cycle cherché. Si ce n'est ni l'un ni l'autre, nous recommençons le pas a.

b) Nous supprimons les derniers arcs de C, jusqu'à ce que le pas a puisse à nouveau être appliquée avec un arc d' non encore utilisé.

Voici l'évolution de c, pour l'exemple de la figure n° 22

a : C = dd4 a : C = dd4d1 a : C = dd4d1d2

b : C = dd4d1 C = dd4

a : C = dd4d3 d3 est adjacent à l'origine de d : d4

Le cycle cherché s'écrit donc +d+d4-d3

III.6. Algorithmes de minimisation en nombre

Nous avons vu, au paragraphe précédent, que, dans le cas où les temps d'exécution des arcs ne sont pas connus au moment du choix des lieux d'implantation des appels, la minimisation en nombre se ramène à la détermination d'un arbre maximal du graphe réduit G'.

Les algorithmes I1 et I2, que nous avons exposés précédemment, permettent de calculer la répartition associée à une telle solution.

Nous allons étudier maintenant quelques uns des algorithmes classiques qui réalisent la construction d'arbres maximum d'un graphe.

III.6.1. Méthode combinatoire [B1]

Soit G un graphe possédant p noeuds et k arcs qui sont d1 d2 ... dk.

Soit E = $\emptyset$

Pour i = 1 à k nous réalisons l'itération suivante : E = E $\cup$ di si di ne constitue pas de cycle avec les arcs déjà dans E.

Cet algorithme peut être considéré comme terminé dès que E possède p-1 arcs.

III.6.2. Méthode booléenne [K4]

Nous attachons à chaque arc du graphe une variable booléenne.

Soit n1 n2 ... np les noeuds du graphe.

Pour chaque noeud, autre que np, nous formons la somme booléenne des variables des arcs adjacents à ni :

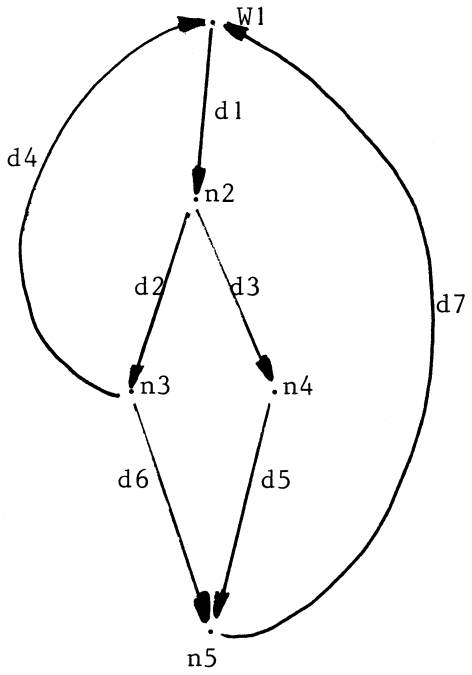
- sous forme directe si l'arc part de ni-i
- sous forme complémentée pour les autres.

Nous formons le produit de ces sommes.

Si des termes possèdent les variables des mêmes arcs sous forme complémentée ou non, nous les supprimons. Les termes restant correspondent aux arbres maximaux du graphe.

EXEMPLE :

Pour le graphe de la figure n° 23, nous associons aux arcs d1d2d3d4d5d6d7, respectivement, les variables booléennes a1a2a3a4a5a6 et a7.



Les facteurs sont :

pour W1 : a'4+a'7+a1

(a'i est le complément booléen de ai)

pour n2 : a'1+a2+a3

pour n3 : a'2+a4+a6

pour n4 : a'3+a5

L'expression booléenne est donc :

(a'4+a'7+a1) (a'1+a2+a3)

(a'2+a4+a6) (a'3+a5)

FIGURE N° 23

En développant, nous obtenons :

~~a'4a'1a'2a'3~~ + a'7a'1a'2a'3 + a'7a'1a'4a'3 + a'7a'2a4a'3 + ~~a1a2a4a'3~~
+ a'4a'1a6a'3 + a'4a2a6a'3 + a'7a'1a6a'3 + a'7a2a6a'3 + a1a2a6a'3
+ ~~a'4a'1a'2a5~~ + a'4a3a'2a5 +a'7a'1a'2a5 + a'7a3a'2a5 + a'1a3a'2a5
+ a'**7a**'1a4a5 + a'7a2a4a5 + a'7a3a4a5 + ~~a1a2a4a5~~ + a'1a3a4a5 + a'4a'1a6a5
+ a'4a2a6a5 + a'4a3a6a5 + a'7a'1a6a5 + a'7a2a6a5 + a'7a3a6a5 + a1a2a6a5
+ a'1a3a6a5

Nous obtenons les arbres maximaux suivants :

d1 d2 d3 d7
d1 d3 d4 d7
d2 d4 d3 d7
d1 d3 d4 d6
d2 d3 d4 d6
d1 d3 d6 d7
d2 d3 d6 d7
d1 d2 d3 d6
d2 d3 d4 d5
d1 d2 d5 d7
d2 d3 d5 d7
d1 d2 d3 d5
d1 d4 d5 d7
d2 d4 d5 d7
d3 d4 d5 d7
d1 d3 d4 d5
d1 d4 d5 d6
d2 d4 d5 d6
d3 d4 d5 d6
d1 d5 d6 d7
d2 d5 d6 d7
d3 d5 d6 d7
d1 d2 d5 d6
d1 d3 d5 d6

Cette méthode risque en fait d'être très longue pour un graphe possédant beaucoup d'arcs. La méthode combinatoire est beaucoup plus rapide et plus facile à réaliser.

III.6.3. Méthode combinée

Nous allons exposer maintenant la méthode que nous avons utilisée pour faire la minimisation en nombre dans les programmes que nous avons écrits.

Soit G' = (X', $\mathcal{U}$). Le graphe réduit représentant un programme particulier.

III.6.3.1. Numération des noeuds

En un premier temps, nous numérotons les noeuds de façon que chacun ait comme prédécesseur un noeud, au moins, de numéro inférieur.

Nous avons déjà vu comment une telle numérotation peut être obtenue en parcourant le graphe à partir du noeud W1 qui sera lui numéroté 1.

Dans la pratique, une telle numérotation s'obtient, nous le verrons, de manière implicite, au moment de la construction du graphe réduit associé au programme analysé.

III.6.3.2. Construction de la répartition correspondant à un minimum en nombre.

Soit R la répartition réelle sur G'.

En suivant l'ordre 2 3 ... p pour les p-1 noeuds de X' - {W1}, nous appliquons successivement sur R, TV2, n2, TV3, n3, ..., TVp,np, les éléments de T tel que :

vi = TVi-1,ni-1 (TVi-2,ni-2(...TV2,n2(R)...)) (d) avec d l'arc qui arrive à ni d'un noeud de numéro inférieur.

Cet algorithme, annule pour **le compte de chaque** noeud de X' - {W1} la valeur associée à l'arc venant d'un noeud de numéro inférieur.

THEOREME V

La construction précédente donne une solution minimale en nombre, c'est-à-dire, D, l'ensemble des arcs, dont les valeurs sont annulées par les applications TV2,n2 TV3,n3 ... TVp,np constitue dans G' un arbre maximal.

DEMONSTRATION

Soit G'D le sous-graphe de G' constitué par les arcs de D.

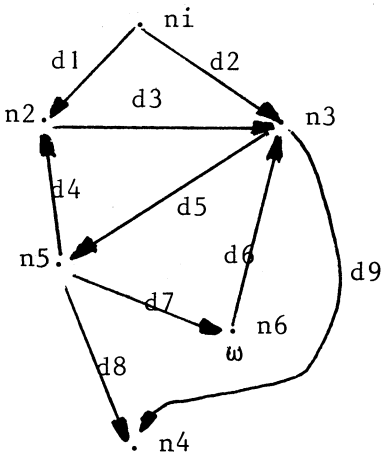
G'D possède p-1 arcs et p noeuds.

C'est un arbre car si nous lui supprimons un arc,nous obtenons p-2 arcs pour p noeuds, c'est-à-dire un graphe qui ne peut pas être connexe.

G'D est un arbre maximal car il possède tous les noeuds de G'.

III.6.3.3. Exemple d'application

Soit le graphe G :



son graphe réduit G' :

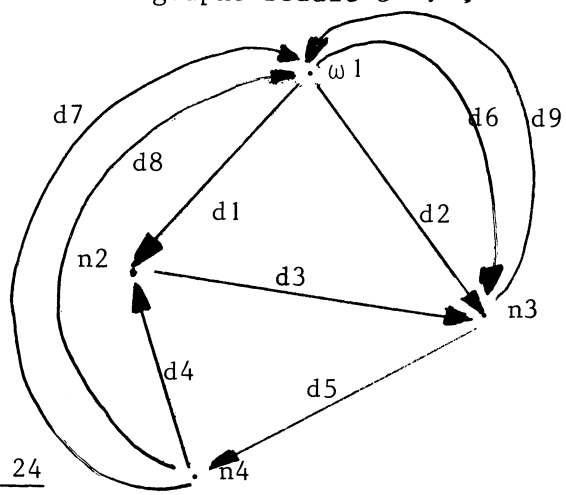


FIGURE N° 24

R la répartition réelle sur G peut être décrite par la suite (t1,t2,t3,t4,t5,t6,t7,t8,t9). Nous avons R(di) = ti

a) Numérotation des noeuds de G'

Nous parcourons les chemins d1 d3 d9 et d1 d3 d5 en numérotant les noeuds au fure et à mesure de leur rencontre. Nous obtenons ainsi la numérotation de la figure n° 24.

b) Construction de la solution minimale en nombre

Nous appliquons successivement :

TV2,n2 avec V2 = t1

nous obtenons comme nouvelle répartition :

(0,t2,t3+t1,t4-t1,t5,t6,t7,t8,t9)

TV3,n3 avec V3=t2

Nous obtenons :

(0,0,t3+t1-t2,t4-t1,t5+t2,t6-t2,t7,t8,t9+t2)

TV4,n4 avec V4 = t5+t2

nous obtenons :

(0,0,t3+t1-t2,t4-t1+t5+t2,0,t6-t2,t7+t5+t2,t8+t5+t2,t9+t2)

La répartition obtenue associe la valeur 0 à trois arcs sur sur 9. C'est bien une solution minimale puisque nous avons autant de valeurs nulles que de noeuds de X' - {W1}. La réduction pour le nombre des appels est donc de 1/3. En général, elle sera plus importante car les noeuds sont de faible degré (2 ou 3). Elle s'approchera de la moitié des appels.

REMARQUE :

Cette dernière méthode est appelée combinée car elle permet de faire en un seul pas ce que les autres font en deux pas :

-- la détermination d'un arbre maximal

-- la construction de la répartition lui correspondant.

En effet, numérotation et construction peuvent être réalisées en même temps : chaque fois que nous rencontrons un noeud non encore numéroté, nous réalisons le pas de la construction lui correspondant.

III.7. Algorithme de minimisation en moyenne

Les méthodes précédemment exposées, ne tiennent aucun compte dans le choix des arcs où seront implantés les appels, du nombre de passages dont ils font l'objet en moyenne à l'exécution. Pour diminuer réellement le temps d'exécution du modèle, nous avons cependant intérêt à mettre les appels de préférence sur les arcs qui sont le moins fréquentés.

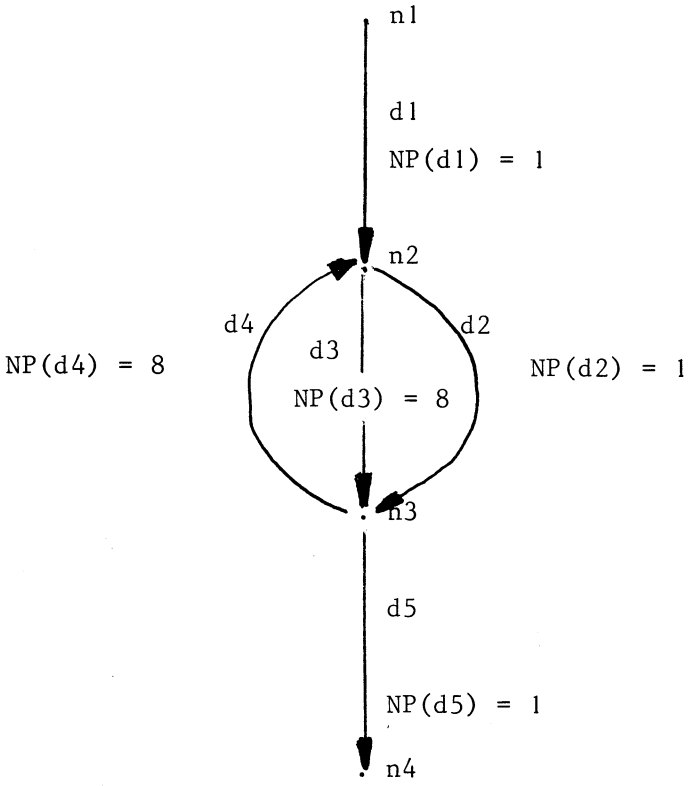


FIGURE N° 25

Notons NP le métrage du graphe G, qui associe à chaque arc, son nombre moyen de passages.

Nous appellerons nombre moyen d'appels de la répartition R sur G, le nombre NR de fois qu'à l'exécution, nous passons, en moyenne, sur un arc d tel que $R(d) \neq 0$, c'est-à-dire sur un arc où se trouve un appel. NR est en fait le nombre moyen d'appels exécutés.

$$NR = \sum_{ni=0}^{\infty} n.P(N=n)$$

avec P(N=n) la probabilité pour que N, le nombre d'appels exécutés soit égal à n.

Soit $C = \{C1\ C2\ \ldots\ Cj\ \ldots\}$ l'ensemble des chemins de la source au puits. (ou de W1 à W1 dans le graphe réduit).

$NR = \sum_{Cj \in C} P(cj).nba(cj)$ avec nba(cj) le nombre d'arcs de Cj où se trouve un appel.

THEOREME VI

Soit R une répartition sur G.

Soit D l'ensemble des arcs d, tels que R(d) = 0.

$(D = R^{-1}(0))$.

Nous avons l'égalité :

$$NR = \sum_{d \notin D} NP(d).$$

DEMONSTRATION

Soit C = {C1 ... Cj ...} les chemins de la source au puits.

$NP(d) = \sum_{Cj \in D} P(cj) \cdot Nd(cj)$ avec Nd(cj) le nombre d'occurrences de d dans Cj.

$$\sum_{d \notin D} NP(d) = \sum_{d \notin D} \sum_{Cj \in C} P(cj) \cdot Nd(cj)$$

on permutte les signes Σ:

$$\sum_{d \notin D} NP(d) = \sum_{Cj \in C} \sum_{d \notin D} P(cj) \cdot Nd(cj) = \sum_{Cj \in C} P(cj) \cdot \sum_{d \notin D} Nd(cj)$$

or nba(cj) le nombre d'appels sur Cj, est égal au nombre d'arcs de Cj n'appartenant pas à D.

$$nba(cj) = \sum_{d \notin D} Nd(cj)$$

donc

$$\sum_{d \notin D} NP(d) = \sum_{Cj \in C} P(cj) \cdot nba(cj) = \underline{NR}$$

CQFD

REMARQUE

Soient ni un noeud du graphe

d1 d2 ... dp les arcs y arrivant

d'1 d'2 ... d'R ceux en sortant

De manière générale nous aurons l'égalité :

$NP(d1) + NP(d2) + \ldots + NP(dp) = NP(d'1) + NP(d'2) + \ldots + NP(d'k)$
(c'est la loi de KIRCHOFF)

Certains noeuds correspondant à un déroutement, peuvent ne pas vérifier cette loi.

Soit G un graphe de programme et G' son graphe réduit. Appliquons à G' l'algorithme de minimisation en nombre exposé au paragraphe précédent et choisissons au pas i l'application TVi, ni qui annule la valeur de l'arc d pour lequel NP(d) est maximum, parmi ceux adjacents à ni qui ne seront plus touchés par les transformations suivantes.

Le choix se fait, par conséquent, entre les arcs adjacents à ni dont l'autre extrémité est soit le noeud W1 soit un noeud de numéro inférieur.

THEOREME VII

Soit R la répartition obtenue, avec la méthode précédente.
R n'est pas forcément une solution minimale en moyenne.

DEMONSTRATION

Le graphe de la figure n° 26 nous fournit un contre exemple.
Nous choisissons pour les noeuds n2,n3,n4,n5 et n6 d'annuler respectivement les valeurs des arcs d1,d2,d5,d7 et d8.
Nous obtenons donc une répartition R telle que :

$NR = 2 + 3 + 7 + 1 + 1 = \boxed{14}$

Appliquons maintenant la transformation TV,n4 avec $V = -R(d6)$. Soit R' la répartition ainsi obtenue, l'arc d5 correspond à nouveau à une valeur non nulle.

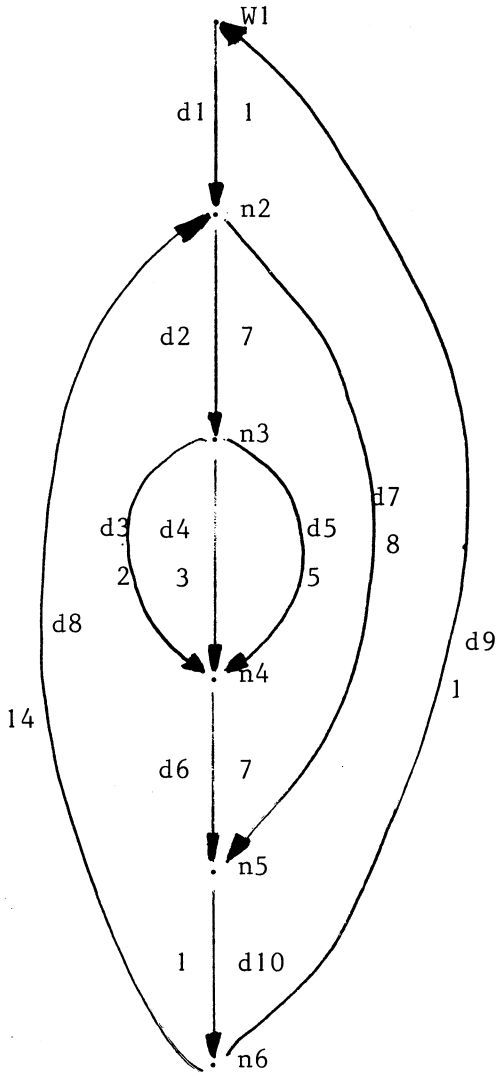


FIGURE N° 26

Les arcs de valeurs nulles sont alors: d1 d2 d6 d7 et d8.

Le nombre moyen d'appels de R' est donc :

NR' = 2 + 3 + 5 + 1 + 1 = 12

Donc : NR' < NR

NR n'est pas minimale.

nous avons noté sur les arcs leurs nombres moyens de passages.

Soit D un ensemble d'arcs supprimables.

Notons N(D) la somme des nombres moyens de passages des arcs de D.

Le problème de trouver une solution minimale en moyenne est équivalent à celui de trouver un ensemble D d'arcs supprimables tel que N(D) soit maximum. Notons Nmax ce maximum. Pour tout arc d, NP(d) > 0 car aucun arc est inaccessible.

Par conséquent, tout ensemble supprimable D, tel que N(D) = Nmax, constitue forcément un arbre maximal.

Une solution minimale en moyenne est également une solution minimale en nombre.

ALGORITHME DE CONSTRUCTION D'UN ENSEMBLE D'ARCS D TEL QUE N(D) = Nmax (A2)

Soient d1 d2 ... dp les arcs du graphe réduit G', classés de façon que NP(di) > NP(di+1).

Au départ, D = {d1}

Pour i = 1 à p nous réalisons l'itération suivante :

-- Si DUdi n'est pas supprimable, nous passons directement au pas suivant :

-- Si DUdi est supprimable, nous faisons :

D = DUdi

Nous pouvons arrêter l'algorithme dès que D possède autant d'éléments qu'il y a de noeuds dans X' - {W1}.

REMARQUE

Etant donné un arbre maximal A et un métrage M, nous appellerons longueur de l'arbre A dans M la somme des valeurs de M correspondant aux arcs de A.

L'algorithme précédent construit par conséquent un arbre maximal D de longueur maximum pour le métrage Np.

C'est en fait le transposé de l'algorithme de KRUSKAL [K5] qui construit lui un arbre maximal de longueur minimum pour un métrage donné.

A partir de l'ensemble d'arcs fourni par cet algorithme et les itérations I1 ou I2, nous obtenons une solution minimale en moyenne.

Pour savoir si au pas i, l'ensemble DUdi est supprimable ou non, il faut voir si le sous-graphe de G' constitué par les arcs de DUdi contient un cycle ou non.

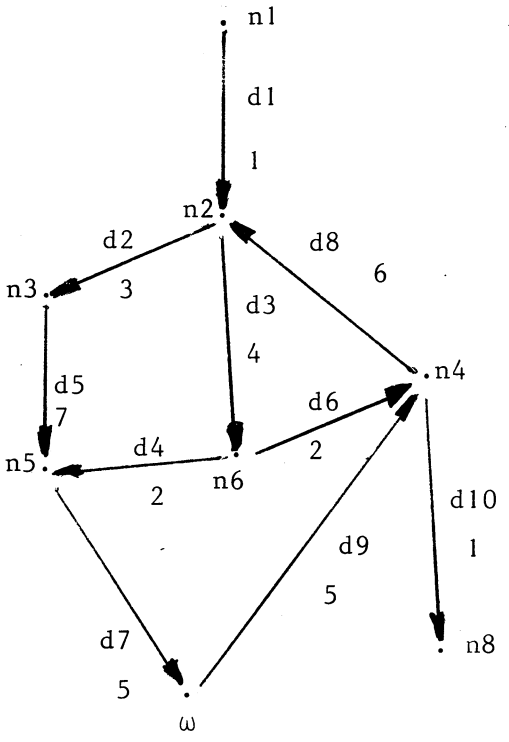
Nous pouvons utiliser, pour celà, les algorithmes classiques de détermination des cycles d'un graphe.

Il n'est cependant pas nécessaire de déterminer tous les cycles pour savoir s'il y en a ou non. De plus, nous savons que l'ensemble D ne possède pas de cycle, c'est-à-dire que G'D le sous-graphe formé par les arcs de D est constitué par un certain nombre d'arbres.

Il suffit donc de savoir si di relie deux noeuds du même arbre de G'D ou non.

Dans le premier cas, il y a un cycle; dans le second, il n'y en a pas.

EXEMPLE D'APPLICATION



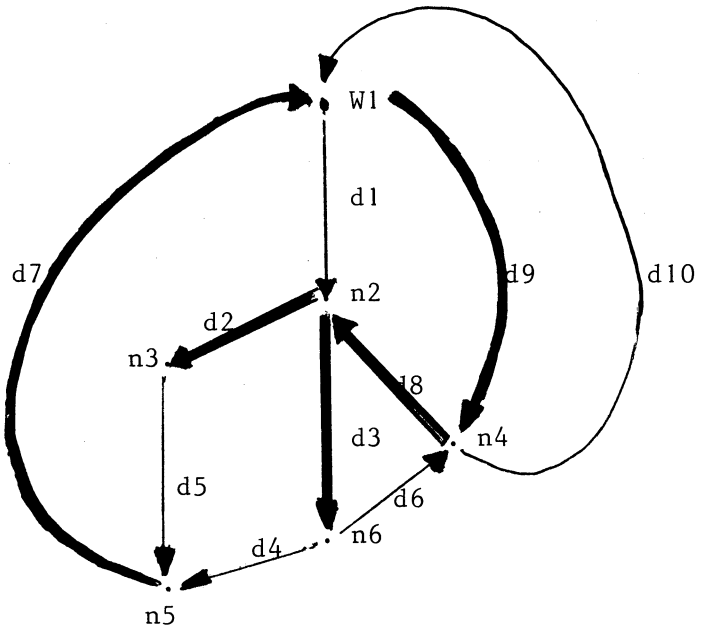
Sur chaque arc du graphe de la figure n° 27 nous avons indiqué son nombre moyen de passages. Soient d8 d9 d7 d3 d2 d5 d4 d6 d1 d10, les arcs ordonnés par valeur décroissante de ces nombres.

Au départ, D = d8.

DU d9 = {d8,d9} il ne contient pas de cycle de G'.

D = {d8,d9}

DU d7 = {d8,d9,d7}



Il ne contient pas de cycle de G'
D = {d2,d9,d7}
DUd3 = {d8,d9,d7,d3}
Il ne contient pas de cycle de G'
D = {d8,d9,d7,d3}
DUd2 = {d8,d9,d7,d3,d2}
Il ne contient pas de cycle de G'.
L'ensemble D = {d8,d9,d7,d3,d2} constitue dans G' un arbre maximal qui correspond à une solution minimale en moyenne.

FIGURE N° 27

Construisons par I2, la répartition R qui correspond à D.

Les cycles fondamentaux sont :

$$\begin{cases} + d1 - d8 - d9 \\ + d4 + d7 + d9 + d8 + d3 \\ + d5 + d7 + d9 + d8 + d2 \\ + d6 + d8 + d3 \\ + d10 + d9 \end{cases}$$

La répartition R est définie par :

$$\begin{cases} R(d1) = t1-t8-t9 \\ R(d2) = 0 \\ R(d3) = 0 \\ R(d4) = t4+t7+t9+t8+t3 \\ R(d5) = t5+t7+t9+t8+t2 \end{cases} \qquad \begin{cases} R(d6) = t6+t8+t3 \\ R(d7) = 0 \\ R(d8) = 0 \\ R(d9) = 0 \\ R(10) = t10+t9 \end{cases}$$

$$NR = NP(d1)+NP(d4)+NP(d5)+NP(d6)+NP(d10)$$
$$= 1 + 2 + 3 + 2 + 1 = 9$$

Il y aura en moyenne 9 appels d'exécutés seulement, au lieu de 30, s'il y en avait un par arc.

IV - APPLICATION A DES PROGRAMMES ECRITS DANS UN LANGAGE PARTICULIER : LP70

IV.1. Quelques notions sur le langage LP70

Nous n'avons pas l'intention de décrire ici complètement le langage LP70. Nous allons seulement essayer de le caractériser par rapport aux différentes classes de langages existants afin de mieux cerner les problèmes particuliers posés par son analyse en vue de la construction d'un graphe de programme.

Pour avoir plus de détails sur la syntaxe de ce langage, il suffit de se reporter aux manuels existants L2 .

LP70 peut être placé à un niveau intermédiaire entre le langage d'assemblage et les langages dits évolués.

Il permet d'utiliser n'importe quelle instruction-machine moyennant une déclaration préalable. L'existence d'un certain nombre d'instructions composées comme IF ou LOOP, lui donne une plus grande souplesse à l'écriture.

Sa structure de bloc permet de localiser l'utilisation des variables déclarées en tête de celui-ci.

Il présente cependant les mêmes avantages que l'assembleur :
-- la compilation est très rapide car il est proche du langage machine,
-- il permet d'adresser au niveau de l'octer,
-- il permet également les adressages indirects.

Tout en gardant les possibilités de l'assembleur, ce langage offre une plus grande maniabilité pour l'écriture des programmes systèmes.

Sans entrer dans les détails de la syntaxe, nous allons étudier les différentes instructions qui forment la structure des programmes écrits en LP70.

Un programme LP70 se présente sous la forme d'un segment principal appelé MAIN qui se trouve inséré entre les mots-clés BEGIN et END. Celui-ci peut appeler des segments externes qui sont compilés séparément.

Lorsqu'un programme utilise un segment externe, celui-ci doit être déclaré avant le BEGIN au moyen de la déclaration EXTERNAL SEGMENT

Par exemple, un segment principal appelant le segment externe READF aura la structure suivant :

```
MAIN
EXTERNAL SEGMENT READF
BEGIN
 }
READF
 {
END.
```

Un segment externe appelé PROG qui utilise READF aura la structure suivante :

```
SEGMENT PROCEDURE PROG
EXTERNAL SEGMENT READF
 }
BEGIN
 {
READF
 }
END.
```

Nous traiterons soit des segments principaux, soit des segments externes.

Chaque segment est constitué d'un certain nombre de blocs. Un bloc est un ensemble de déclarations et d'instructions comprises entre les mots-clés BEGIN et END.

Il délimite une zone particulière du programme.

Nous pouvons distinguer deux types de blocs :

-- le bloc procédure

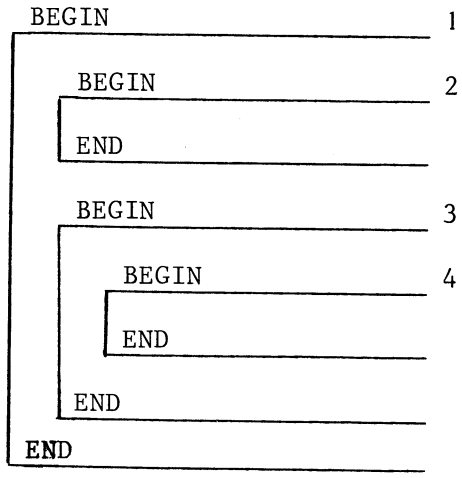
-- le bloc BEGIN - END.

Le premier se différencie du second par le fait qu'il puisse être appelé en différents endroits.

Les blocs peuvent être imbriqués les uns dans les autres. Lorsque deux blocs sont encastrés l'un dans l'autre, le bloc contenant le second est appelé bloc dominant et le bloc contenu dans le premier sous-bloc.

Deux blocs situés au même niveau sont dits indépendants.

EXEMPLE :



Les blocs 2 et 3 sont indépendants. Le bloc 3 est dominant pour le bloc 4 et sous-bloc du bloc 1.

Nous allons étudier les différentes instructions formant l'ossature des programmes :

IV.1.1. Les instructions composées

- L'instruction IF

Elle s'écrit : IF <condition> THEN I1 [ELSE I2]

La partie entre les crochets ([]) est facultative.

Nous n'expliciterons pas la syntaxe d'une condition en LP70. Si cette condition est vrai, on exécute I1, si elle est fausse on exécute la séquence I2, si elle est présente, sinon on passe à l'instruction suivante.

I1 et I2 peuvent être soit une instruction simple, soit une instruction composée, soit un bloc.

Une instruction simple est soit une affectation suivie d'une expression arithmétique, soit un branchement, soit un appel à un sous-programme, soit une instruction machine préalablement déclarée.

- L'instruction CASE

Elle s'écrit : CASE RI/N OF I1 or I2 ... or IN

RI est l'un des 16 registres généraux R0 R1 ... RF. Sa valeur contenue dans RI indique, si elle est comprise entre 0 et l'entier N, le numéro de la séquence à exécuter parmi I1 I2 ... IN.

Si la valeur de RI est négative ou supérieure à N, c'est l'instruction suivant la case qui est à son tour exécutée.

Comme dans le cas du IF, I1 I2 ... IN représentent soit une instruction simple, soit une instruction composée, soit un bloc.

- Les BOUCLES

Il y a trois instructions de boucles qui diffèrent entre elles par le mode de sortie.

Ce sont les instructions LOOP, REPEAT et FOR.

LOOP : elle s'écrit : LOOP b1

b1 est un bloc qui s'exécute un nombre indéfini de fois.

Le seul moyen d'en sortir est de faire un branchement de l'intérieur de b1 vers p'extérieur.

REPEAT : elle s'écrit : REPEAT RI TIMES I1

Si RI contient N, I1 est exécutée N fois. I1 est une instruction simple, une instruction composée ou un bloc.

FOR : Elle s'écrit : FOR RI := EXP STEP K UNTIL B DO I1

EXP : représente ici une expression arithmétique dont la valeur permettra d'initialiser le registre RI qui sert d'indice de boucle. K est un nombre entier qui figure le pas servant à incrémenter à chaque tour RI.

B est la borne supérieure de l'itération.

I1 qui représente une instruction simple, une instruction composée ou un bloc, sera exécutée jusqu'à ce que le contenu du registre RI soit supérieur à l'entier **B.**

IV.1.2. Les instructions simples

- Les instructions de branchement

Outre les instructions machines, BCS, BAL ... etc, qui peuvent être utilisées, en LP70, il existe les instructions suivantes :

GOTO A c'est un branchement inconditionnel à l'étiquette A qui doit être préalablement déclarée en tête d'un bloc englobant l'instruction en question et utilisée suivie du symbole ':' en partie gauche d'une instruction de ce même bloc.

EXEMPLE :

```
BEGIN
LABEL A;
  ≀
GOTO A;
  ≀
A : ..... ;
  ≀
END
```

EXIT N; Après une telle instruction, nous allons nous brancher à la sortie du bloc englobant qui porte l'étiquette N; N peut être un nombre entier ou un nombre hexadécimal précédé du symbole #

Sa présence est facultative.

S'il n'y a pas d'étiquette l'instruction EXIT fait sortir du bloc courant.

EXEMPLE :

```
BEGIN
  ≀
2) BEGIN
  ≀
  LOOP BEGIN
    ≀
    IF COND1 THEN EXIT;
    IF COND2 THEN EXIT2;
    ≀
    END
    ≀
  END
  ≀
END.
```

Le premier EXIT nous branche à la fin de la boucle; le second à la fin du bloc étiqueté 2.

CYCLE N ;

Cette instruction réalise un branchement au début du bloc englobant portant l'étiquette N qui comme pour EXIT est facultative.

CYCLE permet de construire des boucles sans utiliser les instructions prévues à cet effet. Son utilisation est plus performante que celle de l'instruction LOOP qui nécessite un EXIT pour en sortir.

EXEMPLE :

```
BEGIN
  ≀
1) BEGIN
  ≀
  BEGIN
  ≀
  IF COND1 THEN CYCLE
           ELSE CYCLE1;
  END;
  END;
  END.
```

- Les procédures

Elles sont préalablement déclarée par une instruction de ce type :

PROCEDURE NOM; I1.

NOM est le nom de la procédure

I1 est une instruction simple, une instruction composée ou un bloc.

L'appel se fait de la façon suivante :

NOM(P1,P2,...,Pn) avec P1 P2 ... Pn les paramètres qui sont alors chargés dans les registres généraux R0 R1 R2 ... Rn-1. Le registre RF servant à conserver l'adresse de retour, il ne peut y avoir plus de 15 paramètres.

Nous venons, dans ce paragraphe de donner suffisamment de précisions sur la syntaxe de LP70, pour pouvoir comprendre la structure de chaque programme qui est constitué par une succession d'imbrications des instructions que nous avons brièvement décrites.

IV.2. Construction et représentation en mémoire du graphe réduit d'un programme LP70

Nous allons faire correspondre à chaque instruction, un schéma bien précis. Une analyse syntaxique nous permettra de reconnaître les branches du graphe et de construire ce dernier en imbriquant certain de ces différents schémas.

Cette analyse sera faite à l'aide d'un langage particulièrement bien adapté : META5.

IV.2.1. Schémas des instructions

- INSTRUCTION IF

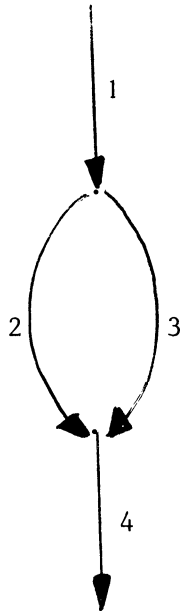
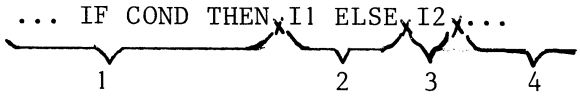


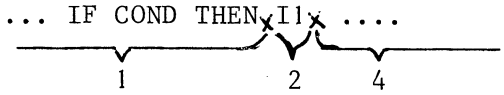
FIGURE N° 28

Comme nous l'avons vu, il y a deux sortes d'instruction IF :

- IF ... THEN ... ELSE ...
- IF ... THEN ...

A toutes deux, nous ferons correspondre le schéma de la figure n° 28. Nous pouvons dans l'écriture de l'instruction délimiter les branches de la manière suivante :





Nous voyons que dans le second cas la branche 3 n'est pas délimitable dans l'écriture du programme. Nous devrons la rajouter dans les tables qui servirons à décrire le graphe. Cet arc ne peut être le support d'un appel aussi, nous le noterons dans une table appelée ARAJ.

- INSTRUCTION CASE :

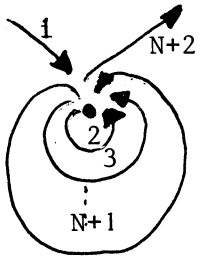
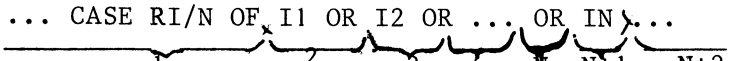


FIGURE N° 29

Soit une instruction CASE comportant N alternatives.

Nous utiliserons pour la représenter le schéma de la figure n° 29, avec la restriction de ne pas utiliser consécutivement deux des arcs parmi ceux notés : 2, 3, 4 ... N+1

Les branches peuvent être délimitées dans son écriture de la façon suivante:



-- LES BOUCLES :

LOOP :

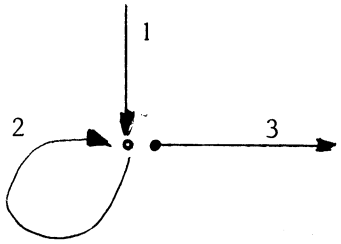
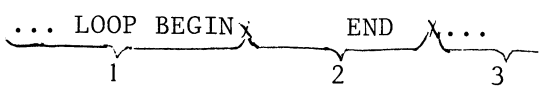


FIGURE N° 30

Nous lui faisons correspnndre le schéma de la figure n° 30.

L'arc 3 est déconnecté. Il sera connecté au reste du graphe par un EXIT ou par un branchement. Les branches se délimitent ainsi :



REPEAT et FOR :

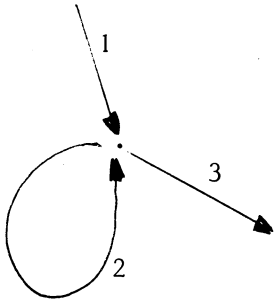
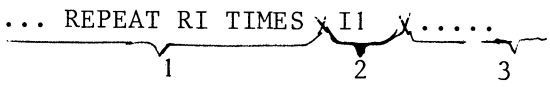
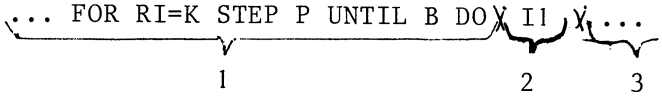


FIGURE N° 31

Ces deux instructions correspondent au même schéma que LOOP avec la différence que la branche 3 est connectée à la sortie de la boucle (figure n° 31).
Les branches peuvent être délimitées comme suit :





-- LA STRUCTURE DE BLOC :

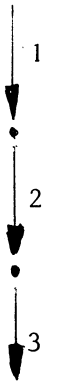
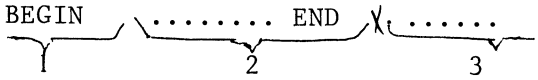


FIGURE N° 32

Il faut mettre un noeud après chaque BEGIN et chaque END pour les instructions EXIT OU CYCLE qui peuvent éventuellement venir se brancher dessus.



-- LES BRANCHEMENTS :

EXIT :

Nous lui faisons correspondre le schéma de la figure n° 33. L'arc numéroté 2 ne pouvant être atteint, est par conséquent inutile. Il n'appartient pas vraiment au graphe réduit.

Pour les reconnaître comme tel dans les tables représentant les différents arcs du graphe, nous lui associerons comme origine un noeud numéroté 0.

Tous les arcs ayant 0 pour origine pourront être supprimés.

Le noeud n figure celui qui suit immédiatement le END après lequel se branche l'instruction EXIT en question.

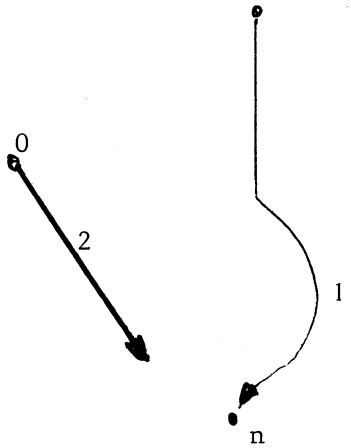


FIGURE N° 33

EXEMPLE : ω1

Le graphe de la figure n° 34 correspond au programme suivant :

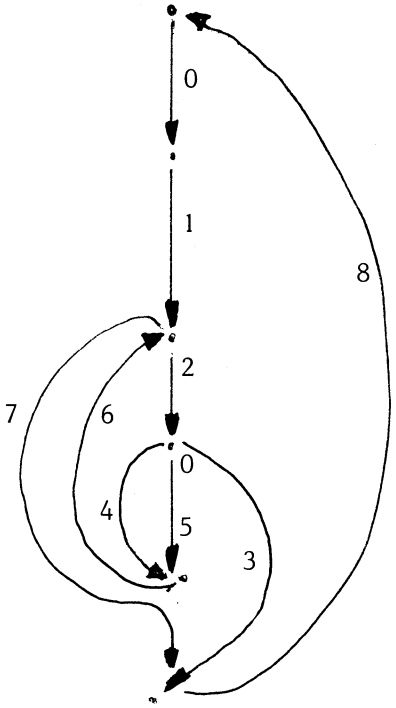


FIGURE N° 34

```
BEGIN                                  }
2) BEGIN                               } 0
LOOP BEGIN                             } 1
IF COND THEN BEGIN                     } 2
EXIT2;                                 } 3
END;                                   } 5
END;                                   } 6
END;                                   } 7
                                       } 8
END.
```

L'arc 4 n'est pas délimité dans l'écriture du programme, il correspond à l'instruction IF sans ELSE.

CYCLE :

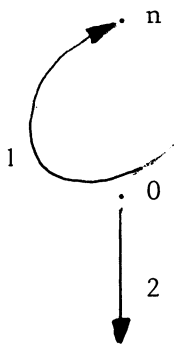


FIGURE N° 35

Nous faisons correspondre à CYCLE un schéma semblable à celui de EXIT.

Le noeud n représente celui qui suit le BEGIN où l'instruction en question va se brancher.

GOTO :

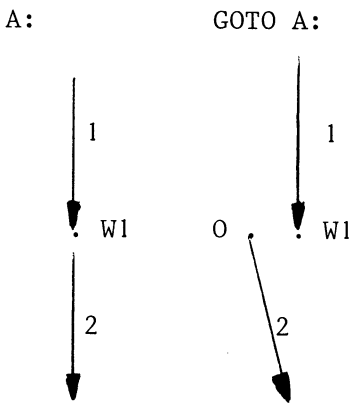


FIGURE N° 36

Nous avons vu qu'en LP70, l'adresse de branchement peut n'être connu qu'à l'exécution. Pour éviter d'importantes erreurs, nous avons choisi de considérer toutes les instructions GOTO et les étiquettes comme des noeuds ω, c'est-à-dire des endroits où le compteur doit contenir le temps réel où en est l'exécution. Nous les représenterons par les schémas de la figure n° 36.

EXEMPLE :

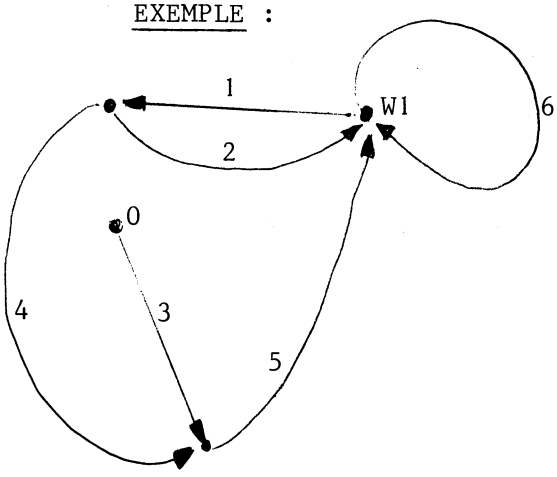


FIGURE N° 37

```
BEGIN
  ----------------------------
LABEL A;                        1
IF COND THEN BEGIN
 X--------------------------
        GOTO A ;                2
          X------------------
        END                     3
          X------------------
                                5
    A:    X------------------
                                6
  X--------------------------
END.
```

Les instructions machines

Les instructions machines réalisant un branchement sont toutes conditionnelles sauf : BCR 0,ETIQUETTE

et : BAL NOM.

La première de ces instructions sera traitée comme l'instruction GOTO ETIQUETTE qui lui est équivalente. La seconde réalise un branchement vers une procédure aussi nous l'assimilerons à un appel de procédure. A tous les autres branchements qui sont conditionnel, nous associerons le schéma de la figure n° 38.

L'arc 2 peut alors être utilisé dans le cas où la condition n'est pas réalisée.

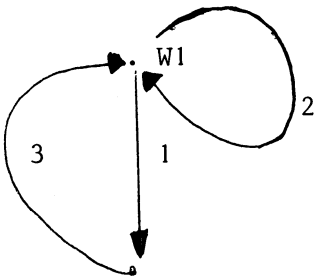
FIGURE N° 38

Les procédures :

La déclaration

Le corps d'une procédure constituant un programme séparé qui peut être appelé de n'importe quel endroit du programme principal, nous l'isolerons du reste du graphe par deux noeuds ω. Les deux parties entourant cette déclaration seront rendues adjacentes.

EXEMPLE :



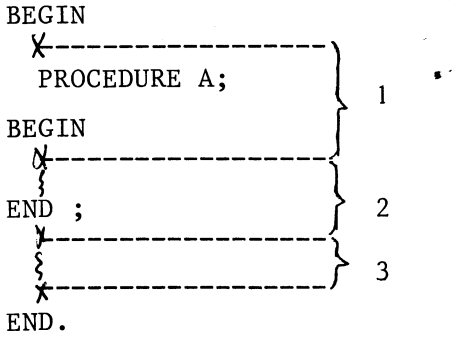


FIGURE N° 39

L'appel

Une procédure pouvant dérouter l'exécution vers un autre programme, nous considérerons tous les appels, comme des noeuds ω.

EXEMPLE :

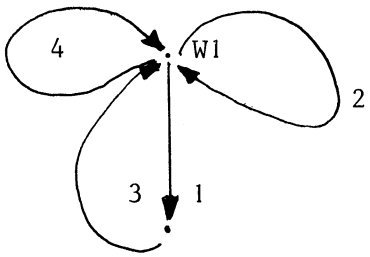
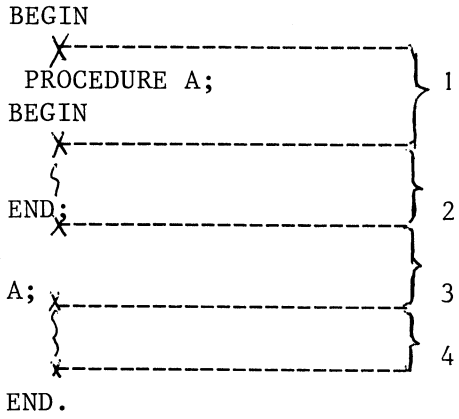


FIGURE N° 40



IV.2.2. Grammaire de schémas

Le graphe réduit, représentant un certain programme LP70, pourra être obtenu par imbrications des schémas élémentaires précédemment exposés. Nous pouvons modifier la grammaire context free du langage pour en constituer une qui pour un programme donné fournira le graphe réduit correspondant [K3].

Nous allons, par exemple, décrire le langage LP70 au moyen de la grammaire G1 qui est la suivante sous la forme normale de BACCHUS [B3]:

(R1) PROGRAMME::= "MAIN" "BEGIN" INSTRUCTION "END."

(R2) INSTRUCTION::= INST/INST ";" INSTRUCTION

(R3) INST::=GOTO/IF/EXIT/CYCLE/LABEL/CASE/FOR/LOOP/REPEAT/BLOC/AUTRINST

(R4) GOTO::= "GOTO" ID

(R5) ID::= un identificateur : nous n'en explicitons pas la syntaxe, pour simplifier cet exposé.

(R6) IF::= "IF" COND "THEN" INSTR/"IF" COND "THEN" INSTR "ELSE" INSTR.

(R7) COND::= une condition : nous n'en explicitons pas la syntaxe

(R8) INSTR::= INST/"BEGIN" INSTRUCTION "END"

(R9) LOOP::= "LOOP" "BEGIN" INSTRUCTION "END"

(R10) BLOC::= "BEGIN" INSTRUCTION "END"

(R11) LABEL::= ID ":"

(R12) CASE::= "CASE" R "/" N "OF" INSTR "OR" SCASE
R représente ici un registre et N un nombre entier

(R13) SCASE ::= INSTR / INSTR "OR" SCASE

(R14) EXIT ::= "EXIT" / "EXIT" N

(R15) CYCLE ::= "CYCLE" / "CYCLE" N

(R16) FOR ::= FOR R ":=" EXP STEP N UNTIL N DO INSTR
EXP représente une expression arithmétique

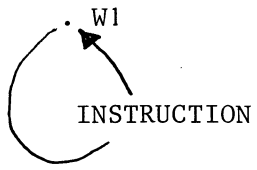
(R17) REPEAT::= "REPEAT" R "TIMES" INSTR

(R18) AUTRINST::= les instructions d'affectation, les expressions arithmétiques ou logiques.

Nous allons modifier ces règles de manière à associer aux primitives des instructions les schémas qui les représentent.

Nous obtenons la grammaire suivante : (G2)

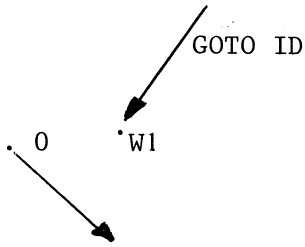
(R1) GRAPHE-PROGRAMME::=



(R2) INSTRUCTION::=INST/INST INSTRUCTION

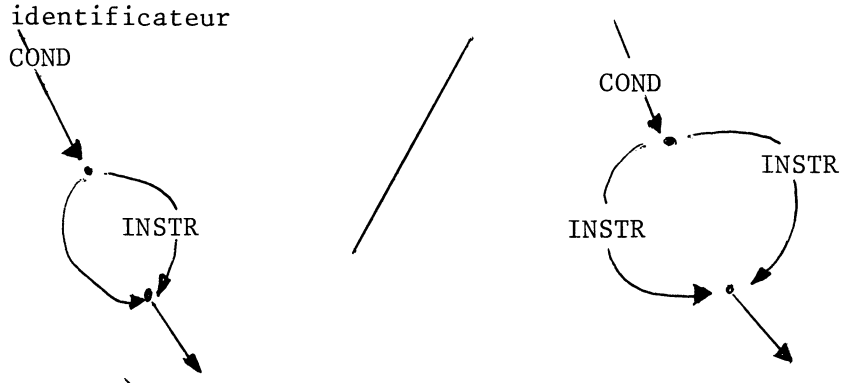
(R3) INST::=GOTO / IF / EXIT / CYCLE / LABEL / FOR / CASE / LOOP / REPEAT / BLOC / AUTRINST

(R4) GOTO::=

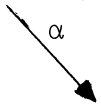


(R5) ID::= identificateur

(R6) IF::=



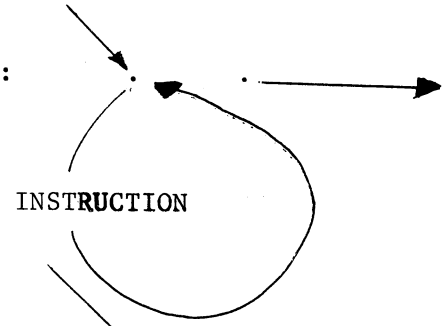
(R7) COND::=



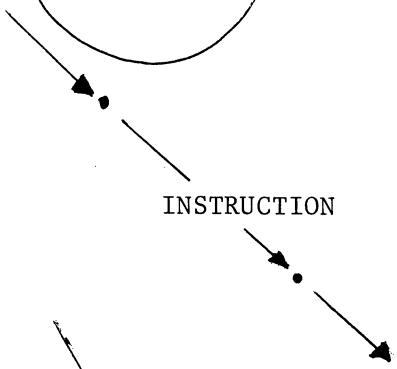
α étant la condition en question

(R8) INSTR::=INST / INSTRUCTION

(R9) LOOP::

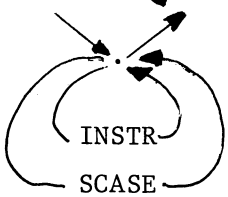


(R10) BLOC::=

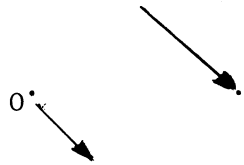


(R11) LABEL::=

(R12) CASE::=

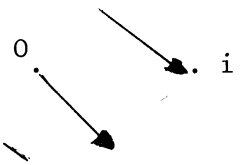


(R14) EXIT::=



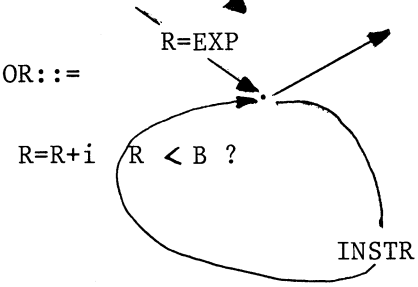
i représente le noeud après le END correspondant au EXIT en question.

(R15) CYCLE::=



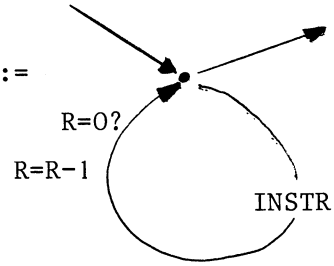
i représente le noeud du BEGIN correspondant au CYCLE en question

(R16) FOR::=



EXP est l'expression d'initialisation de R i
i le pas et B la borne.

(R17) REPEAT::=



(R18) AUTRINST::= I

I l'instruction dont il est question.

Pour voir comment se réalisent les dérivations dans cette grammaire de schémas, nous allons traiter l'exemple suivant :

```
MAIN
BEGIN
R:=0;
LOOP BEGIN R:=2R+1
IF R > 100 THEN EXIT
END;
WRITE(R)
END.
```

Nous pouvons lui faire correspondre l'arbre syntaxique de la figure n° 41.

Nous voyons ainsi que ce programme peut être obtenu par dérivation à gauche, au moyen de l'application des règles de G1 suivantes :

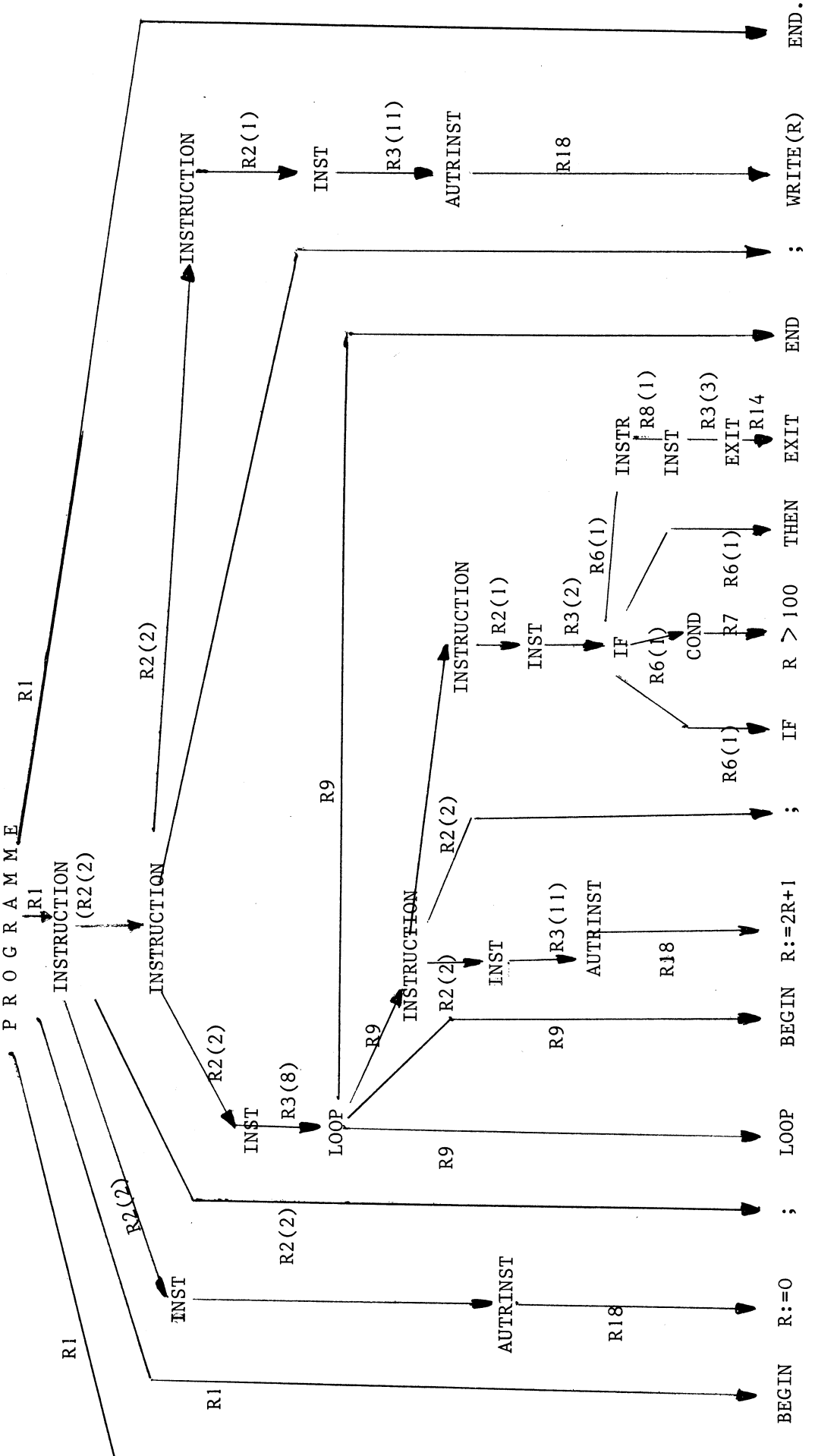
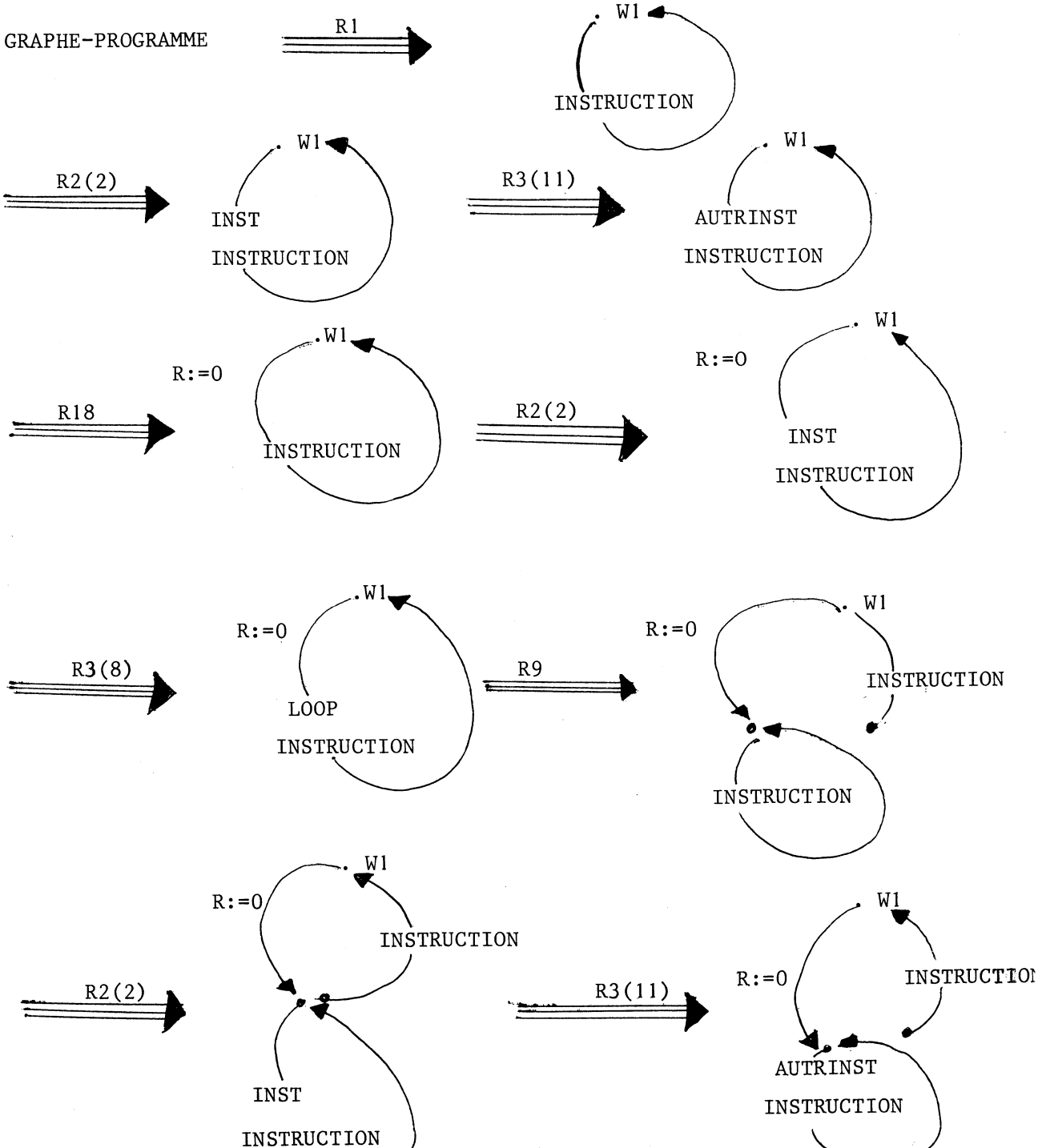

PROGRAMME
R1
INSTRUCTION
(R2(2))
R1
INSTRUCTION
R2(2)
INST
R3(8)
LOOP
R9
INSTRUCTION
R2(2)
INST
R3(11)
AUTRINST
R18
; WRITE(R)
END.
R2(2)
R9
INSTRUCTION
R2(1)
INST
R3(2)
IF
R6(1)
COND
R7
R6(1)
THEN
INSTR
R8(1)
INST
R3(3)
EXIT
R14
EXIT
END
R6(1)
; IF R > 100
R2(2)
R9
INSTRUCTION
R2(2)
INST
R3(11)
AUTRINST
R18
; BEGIN R:=2R+1
R2(2)
; LOOP
R2(2)
INST
R3(11)
AUTRINST
R18
; BEGIN R:=0
R1
R1
AIN

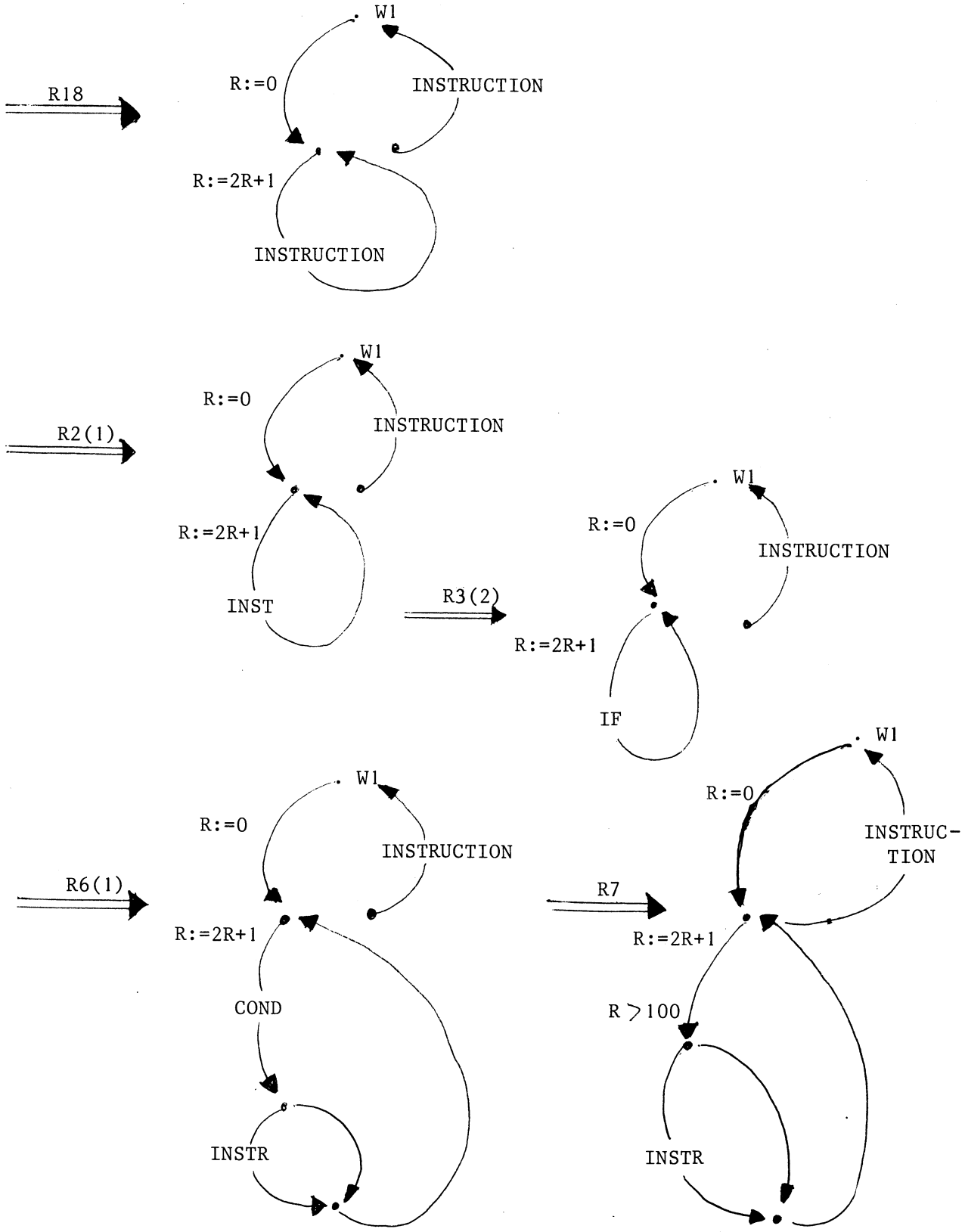
FIGURE 40

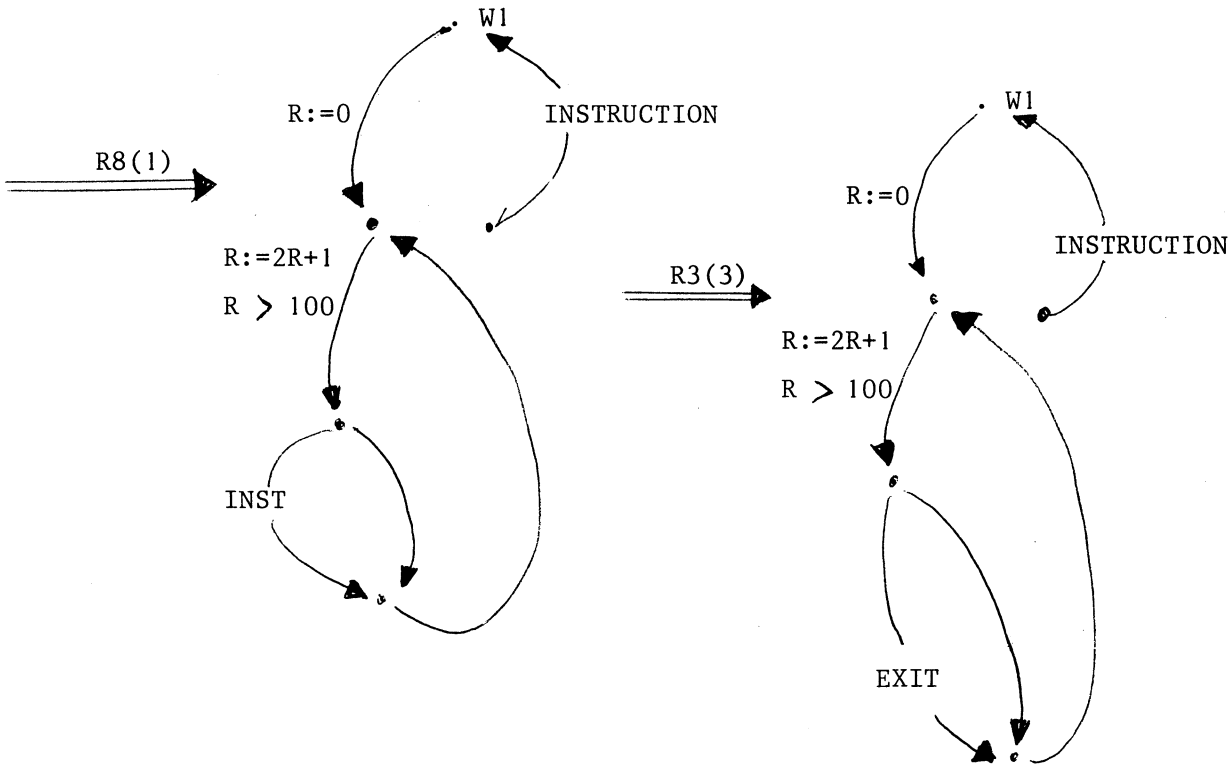
R1, R2(2), R3(11), R18, R2(2), R3(8), R9, R2(2), R3(11), R18, R2(1), R3(2), R6(1), R7, R8(1), R3(3), R14, R2(1), R3(11), R18.

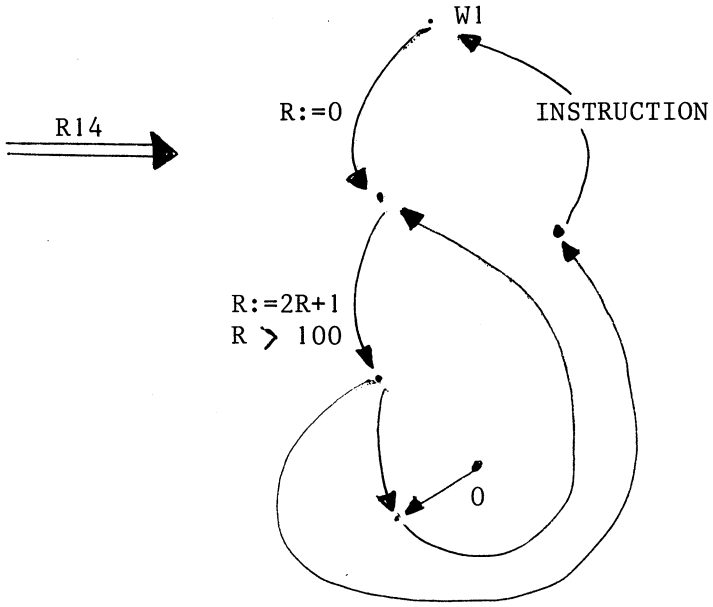
(les chiffres entre parenthèses indiquent, s'il y a lieu, l'alternative de la règle qui a été appliquée).

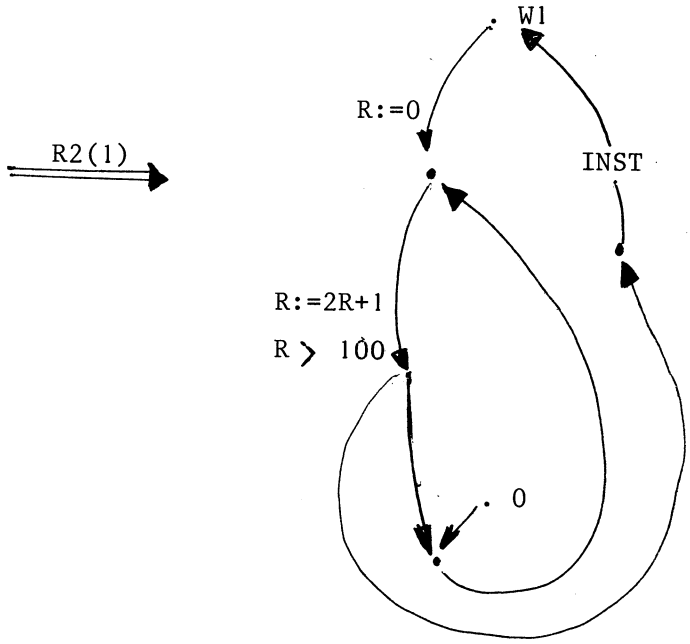
En faisant la même dérivation dans la grammaire de schémas G2, nous obtenons le graphe qui correspond au programme en question.

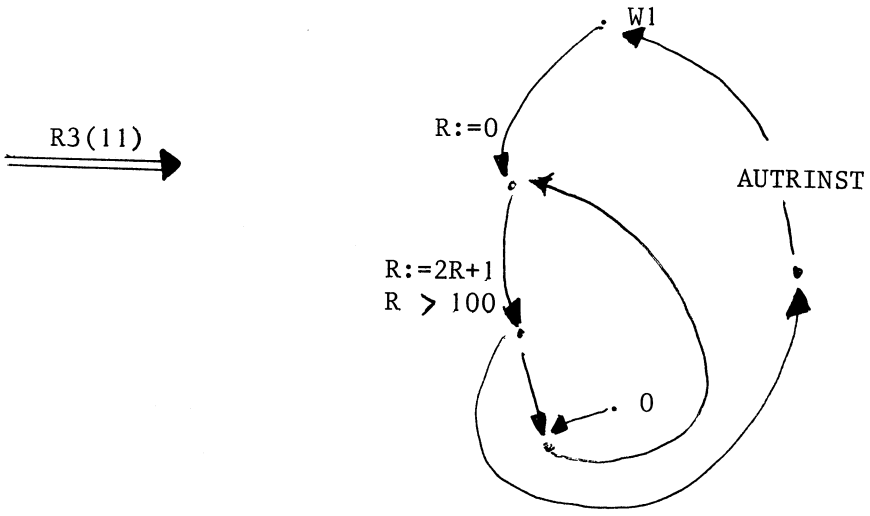


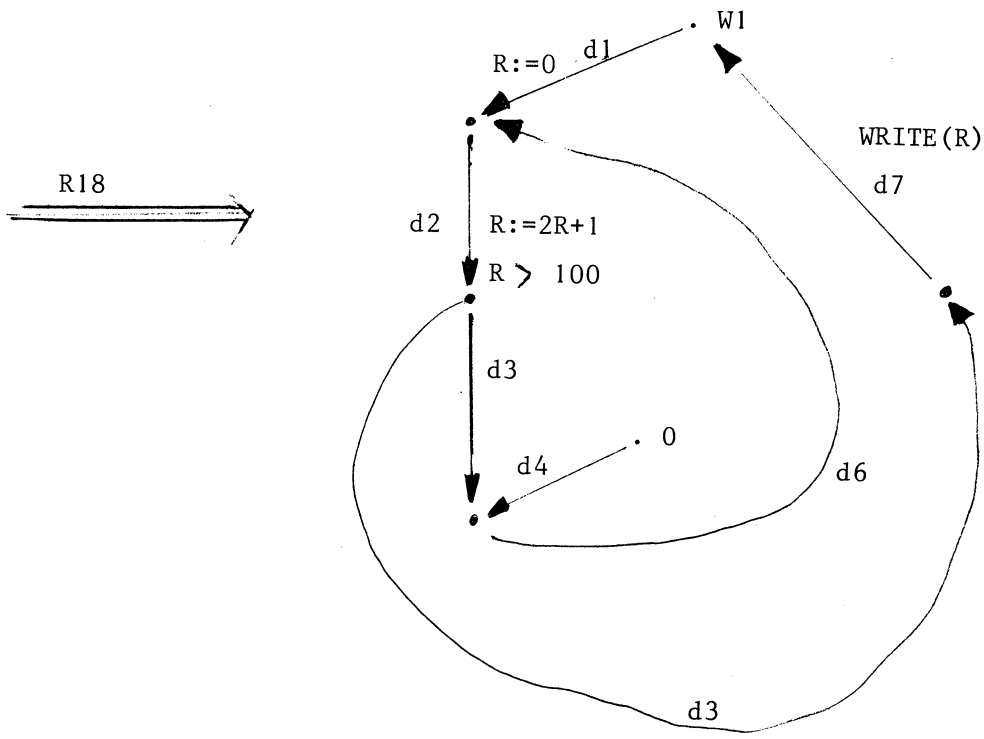

R18
W1
R:=0
INSTRUCTION
R:=2R+1
INSTRUCTION
R2(1)
W1
R:=0
INSTRUCTION
R:=2R+1
INST
R3(2)
W1
R:=0
INSTRUCTION
R:=2R+1
IF
R6(1)
W1
R:=0
INSTRUCTION
R:=2R+1
COND
INSTR
R7
W1
R:=0
INSTRUC-
TION
R:=2R+1
R>100
INSTR


W1
R:=0
INSTRUCTION
R8(1)
R:=2R+1
R > 100
INST
R3(3)
W1
R:=0
INSTRUCTION
R:=2R+1
R > 100
EXIT


W1
R:=0
INSTRUCTION
R14
R:=2R+1
R > 100
0


R2(1)
W1
R:=0
INST
R:=2R+1
R > 100
0


R3(11)
W1
R:=0
AUTRINST
R:=2R+1
R > 100
0



Les arcs du graphe que nous avons obtenu peuvent être délimités dans le programme de la manière suivante :

MAIN ⌐BEGIN R:=0; LOOP BEGIN⌐ R:=2R+1 IF R > 100 THEN⌐ EXIT⌐⌐ ⌐
d1 d2 d3 d4

⌐END.⌐ ⌐WRITE(R)⌐ ⌐ END.
d5 d7

l'arc d5, correspondant à l'instruction IF sans ELSE ne peut être délimité.

Lorsque l'on construit un graphe de programme par une analyse syntaxique, on remonte implicitement dans la grammaire G1, de la chaîne de caractère aux primitives du langage que l'on dérive alors dans G2.

IV.2.3. Représentation du graphe en mémoire

Nous pouvons envisager plusieurs méthodes pour représenter un graphe en machine :

- La méthode matricielle [K4]

Elle consiste à utiliser une matrice dont les lignes et les colonnes représentent respectivement tous les noeuds. Un élément de la matrice est l'ensemble des arcs allant du noeud de sa ligne à celui de sa colonne.

Pour représenter celle-ci, en mémoire, nous pouvons utiliser un tableau T à deux dimensions, dont chaque élément mij est un pointeur vers un autre tableau à une dimension qui contient les numéros des arcs allant du noeud i au noeud j. Le dernier élément de ce tableau de taille variable est 0.

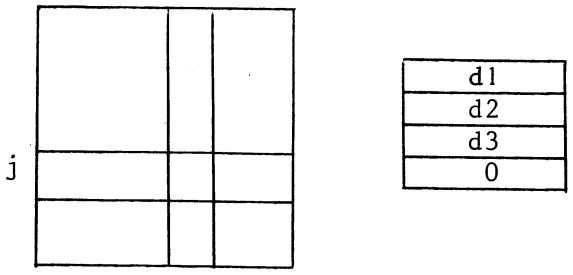


FIGURE N° 41

S'il n'y a aucun arc reliant le couple de noeud en question le pointeur mij à la valeur 0.

L'inconvénient de cette méthode est la place mémoire nécessaire qui est très importante.

Si le graphe est très grand et si le degré des noeuds est faible le tableau T contiendra beaucoup de valeursnulles. Or c'est le cas des graphes des programmes.

- La méthode des pointeurs [K2]

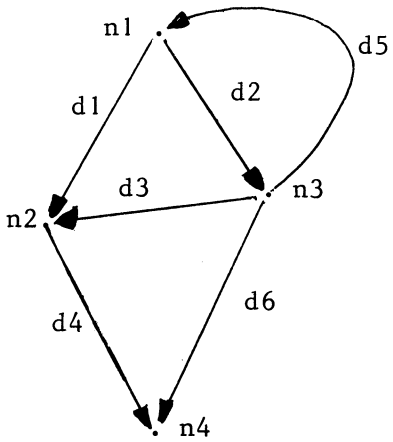
Chaque noeud est représenté par une série de doublet de pointeurs :

A	B

A représente un arc partant du noeud et pointe en conséquence vers un autre noeud.

B pointe vers le doublet suivant du noeud représenté ou contient la valeur 0

EXEMPLE :



Le graphe de la figure n° 42 sera représenté par le schéma de pointeur de la figure n° 43.

FIGURE N° 42

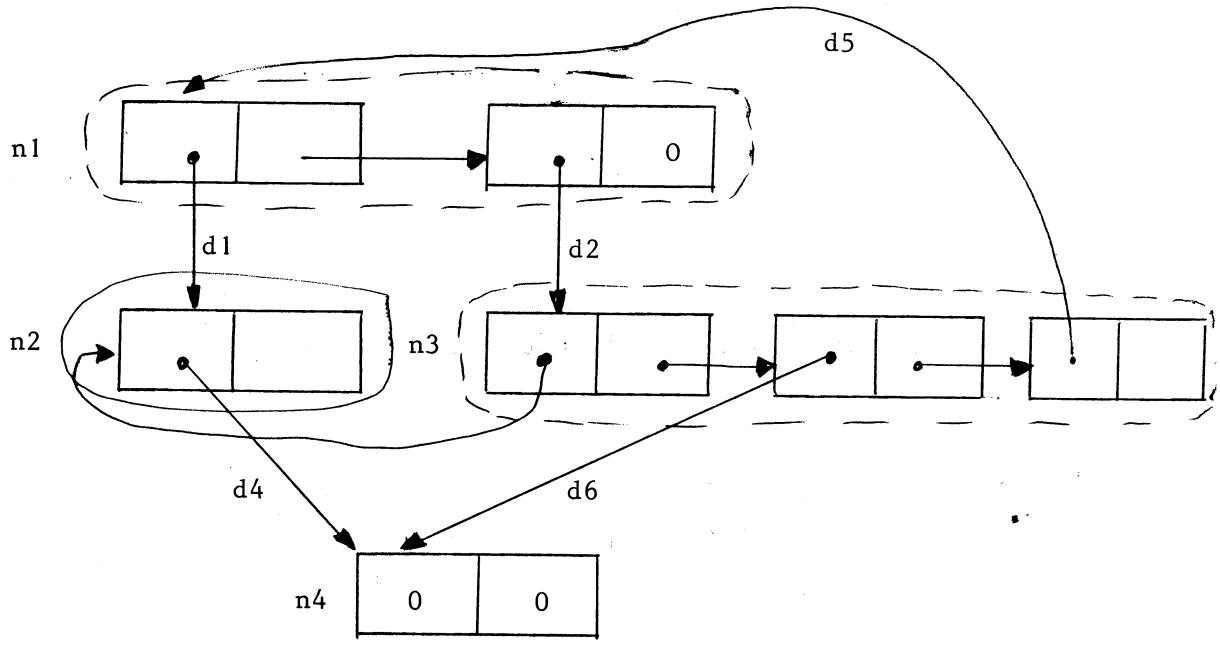


FIGURE N° 43

La représentation matricielle du graphe de la figure n° 42 aurait nécessité 28 emplacements-mémoires, celle-ci en utilise 14.

Elle représente cependant l'inconvénient suivant : si nous voulons obtenir un arc particulier du graphe, nous devons parcourir sa représentation par le schéma de pointeur. Cette méthode n'est pas particulièrement intéressante dans notre cas, car peu maniable.

- Méthode tabulée :

C'est la méthode que nous avons utilisée pour représenter le graphe en mémoire.

Elle utilise deux tables : NOD et NOF
qui sont telles que, étant donnée une numérotation d1 d2 ... dp des arcs du graphe, NOD(i) représente l'origine de di et NOF(i) son extrémité.

EXEMPLE :

Le graphe de la figure n° 42 sera représenté par les deux tables :

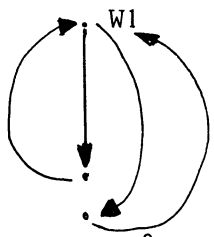
NOD :
n1
n1
n3
n2
n3
n3

NOF :
n2
n3
n2
n4
n1
n4

Nous utilisons ainsi douze emplacements-mémoires soit encore moins qu'avec la méthode des pointeurs.

Cette méthode est intéressante dans notre cas, car l'analyse du graphe se fait séquentiellement suivant l'ordre de rencontre des noeuds et des arcs qui est également leur ordre de numérotation.

Nous pouvons remarquer que les noeuds, numérotés séquentiellement au cours de l'analyse syntaxique, ont tous, excepté W1, un noeud de de numéro inférieur comme prédécesseur.



En effet, si nous ne pouvons accéder à un noeud séquentiellement à partir de l'instruction précédente nous y arrivons par un branchement. Ce noeud est alors W1.

Cette numérotation est donc compatible avec l'application de l'algorithme de minimisation en nombre exposé au chapitre précédent.

IV.2.4. Calcul des temps d'exécution correspondant à chaque arc

Comme, nous l'avons vu, les arcs peuvent être délimités dans le programme écrit en LP70.

Pour calculer les temps d'exécution correspondant à chacun d'eux, en un premier temps, nous insérons dans le programme, des instructions spéciales (CAL3(0,#06)), pour réaliser cette délimitation, les mots-clés BEGIN, END et les points virgules nécessaires pour obtenir un code syntaxiquement correct.

Après compilation, les instructions CAL3 donne un code binaire facilement reconnaissable. Il s'écrit en hexadécimal : 06000000.

Nous évaluons alors les temps de chacun des arcs, numéroté en suivant, à partir du code binaire des instructions qui y ont été générées.

EXEMPLE :

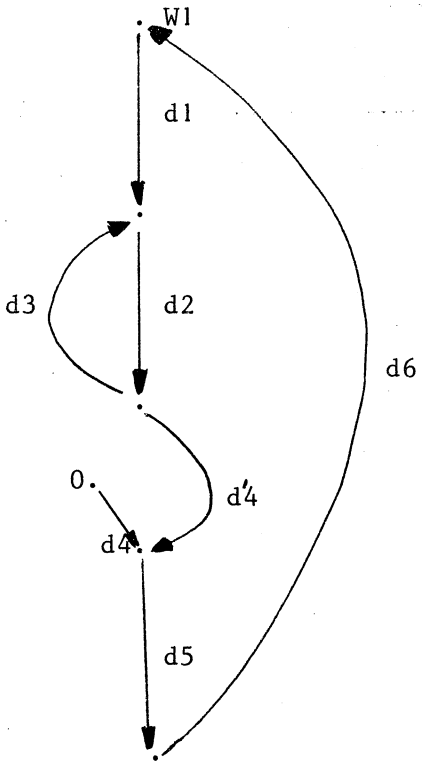
Soit le programme LP70 suivant, obtenu après adjonction des instructions CAL3(0,#06).

```
BEGIN
INSTRUCTION CAL3 #06;
CAL3(0,#06);
R1:=1; R2:=1;
BEGIN                                   d1
CAL3(0,#06);
R2:=R2+1; R1:=R1+R2;
IF R2 < 100 THEN BEGIN                  d2
CAL3(0,#06);
```

```
CYCLE;                                  d3
CAL3(0,##06);
END;                                    d4
CAL3(0,##06);
END;                                    d5
CAL3(0,##06);
WRITE(R1);                              d6
CAL3(0,##06);
END.
```

Ce programme correspond au graphe réduit de la figure n°45.

Après compilation, il donne la séquence d'instructions-machines suivante :




```
                CAL3,0   ##06
                LI,R1    1
d1              LI,R2    1
          E1    CAL3,0   ##06
                AI,R2    1
                AW,R1    R2
d2              CI,R2    100
                BCR,7    E2
                CAL3,0   ##06
d3              BCR,0    E1
                CAL3,0   ##06
d4        E2    CAL3,0   ##06
d5              CAL3,0   ##06
                LW,R0    R1
d6              BAL,RF   WRITE
                CAL3,0   ##06
                END.
```

FIGURE N° 45

REMARQUES :

-- Il existe des arcs de temps nuls, ne correspondant à aucune instruction du programme. Nous pouvons purement et simplement les supprimer en confondant leurs noeuds d'origine avec leurs noeuds d'extrémité.

Celà revient au même pour le nombre d'appels restant après la minimisation puisque pour chaque arc supprimé, nous enlevons également un noeud.

-- L'arc d'4 n'est pas délimitable puisqu'il figure, dans l'instruction IF THEN, le branchement à l'instruction suivante.

Il correspond également à aucune instruction générée, mais représente simplement la possibilité de se brancher après le IF.

Cet arc ne correspondra à aucun élément de la table des temps. Nous l'avons noté, pour en tenir compte, dans une table spéciale appelée ARAJ au moment de la construction du graphe.

IV.2.5. Utilisation du langage META5 [M4]

Créer essentiellement pour décrire les compilateurs, ce langage offre des possibilités de manipulations intéressantes, sur la syntaxe d'un programme écrit dans code particulier.

Il permet, en fait, de réaliser une analyse syntaxique descendante, en suivant la chaîne d'entrée de gauche à droite et de faire certains traitements en conséquence.

Pour celà, nous utilisons, comme mode de description la forme normale de BACCHUS, dans laquelle nous pouvons insérer un certain nombre de procédures qui réalise le traitement correspondant à la chaîne analysée.

Cela fonctionne de manière analogue à la machine de KNUTH [K6], avec en plus la possibilité de retourner en arrière dans l'analyse de la chaîne d'entrée.

La grammaire utilisée ne doit posséder aucune règle récursive à gauche. C'est-à-dire, il ne doit y avoir aucune règle de la forme :

S::= S w1/w2/.../wn

Une telle règle serait testée indéfiniment sur le caractère analysé.

Nous n'entrerons pas dans le détail de la description de cet outil, mais nous allons cependant donner un exemple d'utilisation afin d'en apprécier les avantages et les inconvénients.

Un programme META5 est formé d'un ensemble de règles syntaxiques qui peuvent s'appeler les unes les autres.

Une de ces règles est désignée, au départ, comme étant la règle principale. Le programme se termine lorsque l'analyse et l'exécution correspondant à cette règle se sont achevées.

Chaque règle est formée d'une suite d'alternatives qui sont testées dans leur ordre d'écriture. Les alternatives sont séparées par le symbole "/".

Si aucune n'est vrai, la règle retourne la valeur FAUX.

Nous disposons d'un certain nombre de primitives pour reconnaître des chaînes particulières dans le programme analysé.

EXEMPLE :

.CHAR	Elle reconnaît un caractère quelconque
.NUM	Elle reconnaît un nombre entier.

L'opérateur moins (-) permet de tester l'absence d'une condition : -.NUM est vrai si la chaîne de caractère qui se trouve en entrée ne commence pas par un nombre entier.

Un pointeur indique continuellement le caractère où en est l'analyse de la chaîne d'entrée. Chaque fois qu'un caractère est reconnu le pointeur se trouve incrémenté de un : nous passons en fait au caractère suivant.

Lorsqu'une règle retourne avec la valeur FAUX, il y a ce qu'on appelle un retour arrière ("BACK UP") : tous les pointeurs sont remis à la valeur qu'ils avaient avant d'entamer l'analyse correspondant à cette règle.

Nous pouvons déclarer en tête du programme, un certain nombre de variables de types différents (valeurs, chaîne de caractères, ...) sous forme de piles, tableaux, variables simples.

De plus, il existe une pile spéciale notée * qui sert d'accumulateur. Pour référencer un rang inférieur au sommet, il y a deux écritures possibles : - *(i)

- * -i

i étant un nombre entier qui indique la ième variable à partir du sommet

Si nous référençons une pile sans utiliser d'index, c'est forcément l'élément du sommet qui est pris en compte puis il est supprimé.

Nous allons, pour mieux comprendre toutes ces notions, étudier un exemple de programme META5 et faire en quelque sorte sa trace à l'exécution.

Celui-ci va analyser un programme LP70 afin de construire la table des identificateurs et d'associer à chacun d'entre eux, le nombre de fois qu'ils ont été utilisés.

Ce programme s'écrit :

```
LIGNES
   1          META5 EXEMPLE MAIN
   2          .TABLE ID
   3          .ITEM NOM*B;
   4          .ITEM NOMBRE*V;;
   5       MAIN=$(MOTCLE / IDENTIFIER / .SKIP);
   6       IDENTIFIER = .LETTER $ (.LETTER /.DIGIT) .CONCAT
   7                    (.FIND(*,NOM) .AOR(NOMBRE(*))
   8                    / .ENTER(ID) .PUT(*-1,NOM(*-O))
   9                    .PUT(1,NOMBRE(*)) .POP(*));
  10       MOTCLE = ('BEGIN' / 'END' / 'WORD' / ... etc...);
  11       .END
```

Explication des différentes lignes du programme précédent

LIGNE 1 :

C'est le début du programme META5. Le premier identificateur, qui suit le mot-clé META5, représente le nom du programme. Le second identificateur indique quant à lui, le nom de la règle principale, c'est-à-dire celle par laquelle le programme commence son exécution.

LIGNES 2,3 et 4 :

Elles déclarent une table nommée ID composée par deux colonnes : NOM et NOMBRE

La première de type B, (chaîne de caractères), servira à mettre les noms des différents identificateurs rencontrés.

La seconde de type V, (valeur), recueillera pour chacun le nombre d'occurrences dans le programme analysée.

LIGNE 5 :

La règle MAIN est la règle principale du programme. Le symbole $ indique que ce qui suit entre parenthèses peut être trouvé entre 0 et l'infini de fois dans la chaîne analysée.

. SKIP est une primitive du langage qui sert à sauter purement et simplement le caractère courant de la chaîne d'entrée.

LIGNES 6,7,8 et 9 :

IDENTIFIER est la règle qui permet de reconnaître et de traiter les occurrences des identificateurs. Elle les met dans la table. Si elles y sont déjà, elle se contente d'incrémenter de 1 l'élément de NOMBRE correspondant.

Les primitives utilisées sont les suivantes :

.LETTER : elle permet de reconnaître les vingt six lettres de l'alphabet

Si nous avons bien comme caractère courant une lettre, elle sera mise au sommet de l'accumulateur avec le type B. Le pointeur de la chaîne analysée est alors incrémenté de 1.

Si ce n'est pas le cas, la valeur FAUX est retournée.

.DIGIT : elle permet de reconnaître les chiffres 0 1 2 ... 9.

Elle se déroule suivant le même processus que .LETTER.

.CONCAT : Elle réalise, s'il y a lieu, la concaténation des variables de type B qui se trouve au sommet de l'accumulateur. Le résultat est de type C, c'est-à-dire une chaîne de caractères qui ne peut pas être concaténée à son tour.

.FIND : Elle permet de trouver dans la table indiquée par le deuxième argument ce qui se trouve dans le premier. Si la table est indexée, l'analyse débute seulement à partir du rang indiqué.

Le rang de la première occurrence de la variable cherchée, est mise, s'il y a lieu, au sommet de l'accumulateur. S'il n'y a aucune occurrence de cette variable dans la table, la valeur FAUX est retournée.

.AOR : Elle rajoute 1 à la valeur de la variable qui lui sert d'argument.

.ENTER : Nous avançons de 1 rang, dans l'utilisation de la table dont il est question. La valeur du rang est alors placée sur l'accumulateur. Nous ne pouvons pas utiliser les éléments de la table d'indice supérieur à celui fournit par le dernier .ENTER. Nous sommes obligés d'utiliser cette primitive chaque fois que nous rajoutons un nouveau identificateur à la table ID.

.PUT : Elle met dans la variable indiquée par le second argument la valeur du premier.

.POP : Elle sert à diminuer de 1 la taille d'une pile, en enlevant l'élément qui se trouve à son sommet.

LIGNE 10

La règle MOTCLE permet de reconnaître les mots propres au langage analysé et qui ne doivent pas être pris pour des identificateur Nous ne les avons pas tous écrits, pour simplifier la présentation.

Pour voir la façon dont se déroule l'exécution de ce programme META5, nous allons en étudier une application au programme LP70 suivant :

```
BEGIN
WORD A;
A:=1;
WORD BA2;
BA2:=A*A;
END.
```

Nous pouvons voir que ce programme contient deux identificateurs A et BA2 qui sont utilisés respectivement 4 et 2 fois.

L'analyse commence par l'application de la règle principale : MAIN.

MOTCLE reconnaît le BEGIN du début puis WORD, en retournant les deux fois avec la valeur VRAI. Le pointeur de chaîne est alors positionné sur A;... La règle MOTCLE retourne cette fois avec la valeur FAUX. Nous essayons alors la règle IDENTIFIER. La primitive .LETTER met A sur le sommet de l'accumulateur et les primitives .LETTER et .DIGIT qui sont alors testées retournent FAUX sur le point-virgule.
.FIND retourne FAUX, A ne se trouvant pas encore dans la table ID.
.ENTER permet d'avoir l'indice 0 sur le sommet de l'accumulateur. Nous remplissons alors le premier rang de la table ID par le NOM, A et le NOMBRE 1 :

ID	NOM	NOMBRE
	A	1

Nous revenons de la règle IDENTIFIER avec la valeur VRAI. Nous recommençons donc à essayer les alternatives de la règle principale. Le pointeur est alors positionné sur le point-virgule.

Celui-ci n'étant ni un mot-clé ni une lettre, nous exécutons un .SKIP pour le sauter.

Nous avons alors en entrée : A:=1 ...

Le A sera reconnu comme un identificateur, mais cette fois-ci dans la règle IDENTIFIER, la primitive .FIND donne sur l'accumulateur l'indice 0 qui est le rang de A dans ID. NOMBRE(0) est alors incrémenté de 1.

Les caractères :=, 1 et ; seront sautés par .SKIP. WORD sera reconnu comme un mot-clé et BA2 le second identificateur par IDENTIFIE Je ne poursuivrai pas davantage la trace de cette exécution dont le mécanisme apparaît déjà clairement dans les lignes précédentes.

A la fin, nous avons la table :

ID

NOM	NOMBRE
A	4
BA2	2

REMARQUES SUR META5

Le principal avantage de ce langage est sa facilité d'écriture et d'utilisation. Nous pouvons, en effet, avec META5 écrire en quelques lignes un programme pour modifier de manière systèmatique certaines parties d'une chaîne de caractères, syntaxiquement reconnaissable.

D'autre part, nous pouvons lui reprocher un temps de compilation et d'exécution relativement important. Cela est dû à la complexité des algorithmes qui sont ainsi générés.

L'analyse de 43 cartes LP70 condensés, en vue de la construction du graphe réduit correspondant demande à peu près 50 secondes.

IV.2.6. Programmes de construction du graphe réduit

Il est aisé de construire le graphe réduit d'un programme LP70 en l'analysant par un programme META5 reconnaissant les branches au moyen d'une description sommaire de la syntaxe de ce langage comme celle du paragraphe IV.2.2.

Nous n'allons pas ici entrer dans le détail de l'écriture d'un tel programme.

Nous avons vu que pour calculer les temps d'exécution des arcs, il faut rajouter dans le code du programme un certain nombre d'instructions spéciales (CAL3(0,#06)). Celà se fait également au moyen d'une analyse syntaxique. Il en est de même lorsque nous implantons des instructions pour mesurer les nombres de passages à l'exécution par chacune des branches. Nous avons intérêt à faire le maximum de chose dans la même analyse. Aussi nous avons écrit trois programmes différents en META5 qui tous construisent les tables du graphe.

Les trois programmes META5 sont :

GRAPHEO : Il analyse le programme LP70 dont les commentaires ont été préalablement supprimés. En sortie, il fournit d'une part le code où sont rajoutés les instructions CAL3, et d'autre part, les tables du graphe présentées de la manière suivante :

```
NNOEUD = ..........      (nombre de noeuds)
NARC = ............      (nombre d'arcs)
NOD :
........
NOF :
........
ARAJ :
.......
ARNU :
.......
FIN
```

ARNU contient pour le compte de chaque noeud, le numéro du premier arc rencontré, allant d'un noeud déjà numéroté à celui-ci. Ce sont les arcs dont nous annulerons les valeurs dans la table des temps.

Le mot FIN indique au programme qui traitera ces données que nous réalisons une minimisation en nombre seulement et que les nombres moyens de passages des arcs ne seront pas utilisés.

GRAPHE 1 :

Il analyse en entrée, le programme déjà additionné des CAL3. Il sort les mêmes tables que GRAPHE0 pour représenter le graphe.

GRAPHE 2 :

Il analyse en entrée le programme additionné des CAL3. Il fournit en sortie d'une part le code avec les instructions pour la mesure des nombres de passages par les arcs, d'autre part les tables du graphe, le mot NMPAS remplaçant le mot FIN. Il indique que nous utiliserons les nombres moyens de passages et ferons une minimisation en moyenne des appels.

Nous allons voir quelques unes des difficultés que pose l'écriture de ces programmes :

-- L'adjonction des instructions CAL3 :

Le code obtenu en rajoutant ces instructions doit être, nous l'avons vu, syntaxiquement correct puisqu'il est destiné à être compilé.

Celà pose un certain nombre de problèmes. Notamment les branches des IF ou CASE comportant une seule instruction doivent être remplacées par un bloc contenant le CAL3 et l'instruction en question.

Exemple :

```
IF X=1 THEN Y:=2 ELSE Z:=3; doit être remplacé par :
    IF X=1 THEN BEGIN CAL3(0,#06); Y:=2 END;
    ELSE BEGIN CAL3(0,#06); Z:=3 END;
```

Une deuxième difficulté provient de l'adjonction des point-virgules.

En effet, en LP70, chaque instruction doit être suivie d'un point-virgule, excepté si elle précède les mots-clés OR ou ELSE.

BEGIN n'est jamais suivi d'un point-virgule et il est facultatif devant un END.

Aussi, lorsque nous rajoutons les instructions CAL3, nous devons tenir compte du contexte afin de respecter les règles de syntaxe du langage.

-- La construction des tables

Les tables NOD, NOF et ARAJ se construisent facilement en suivant l'analyse syntaxique du programme.

Pour la table ARNU nous devons faire en sorte que tous les arcs de ARAJ s'y trouvent puisque ces arcs non délimitables dans le code du programme ne peuvent être le support d'aucun appel.

De plus, comme nous voyons dans le cas de la figure n° 46 lorsque pour le compte d'un noeud le choix se fait sur plusieurs arcs, nous devrons autant que possible éviter de mettre dans ARNU un arc provenant du noeud 0 car cet arc qui est inaccessible ne sera de toute façon le support d'aucun appel. Le minimum ne serait alors pas forcément atteint.

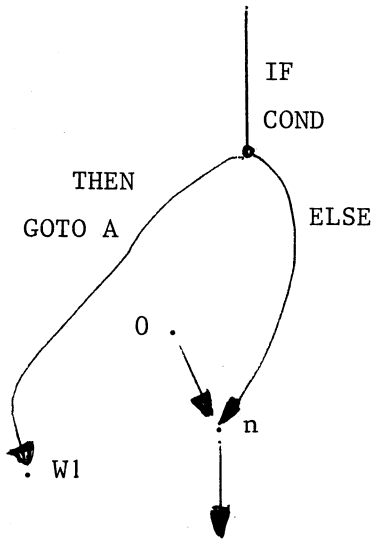


FIGURE N° 46

-- L'adjonction des instructions permettant de mesurer les nombres de passages des arcs

Nous devons tout d'abord déclarer en tête du programme, si et seulement si ces déclarations n'existent pas déjà dans le code analysé, les segments externes CLOSEF, OPENF et WRITEF que nous utilisons pour sortir les mesures sur un fichier, les instructions machines MTW, LCF et STCF servant à faire les incrémentations, un tableau de taille suffisante (NMPAS) destiné à recueillir les nombres de passages des arcs que nous pouvons délimiter dans le code et enfin un mot (LPSVCC) où nous stockerons le code condition pour le restaurer lors de chaque utilisation de l'instruction MTW. Pour éviter les doubles déclarations, nous pouvons imaginer deux procédés :

- Après avoir analyser toutes les déclarations, nous rajoutons seulement celles qui n'y sont pas.
- Nous rajoutons d'office au début. Toutes ces déclarations, puis nous les supprimons, s'il y a lieu, lorsque nous les rencontrons dans l'analyse du programme.

Nous supposons que NMPAS et LPSVCC ont des noms assez particuliers pour qu'il n'y ait que peut de chance de rencontrer ces identificateurs dans un programme.

Pour les segments externes dont les déclarations précèdent le premier BEGIN, nous pouvons utiliser aisément la première méthode. Pour les déclarations des instructions machines qui peuvent apparaître n'importe où dans le code du programme, entre le premier BEGIN et END., il est plus facile d'utiliser la seconde méthode.

En tête de chaque arc, nous rajoutons la séquence :

```
STCF(3,LPSVCC); MTW(1,NMPAS(I)); LCF(3,LPSVCC)
```

I est le numéro de l'arc

STCF stocke le code condition

LCF restaure ce même code condition.

A la fin du programme, nous générons un bloc où se trouvent les déclarations et les instructions nécessaires à la sortie du tableau NMPAS sur un fichier.

IV.3. Minimisation des appels a partir des tables du graphe

La minimisation en nombre ou en moyenne des appels, consiste à modifier la table des temps des arcs du graphe en annulant le plus grand nombre d'éléments.

Celà est fait par un programme écrit en LP70 (MODTEM) Il suffira alors de rajouter des appels de remise à jour seulement sur les arcs qui correspondent dans la table des temps, à une valeur qui n'est pas nulle. La remise à jour consiste en fait à ajouter au compteur la valeur en question.

Suivant si les tables du graphe sont suivies du mot FIN ou non, le programme MODTEM réalise une minimisation en nombre ou une minimisation en moyenne des appels. Dans le second cas, il utilise les nombres moyens de passage par les arcs.

MODTEM peut être décrit par l'organigramme de la figure n° 47.

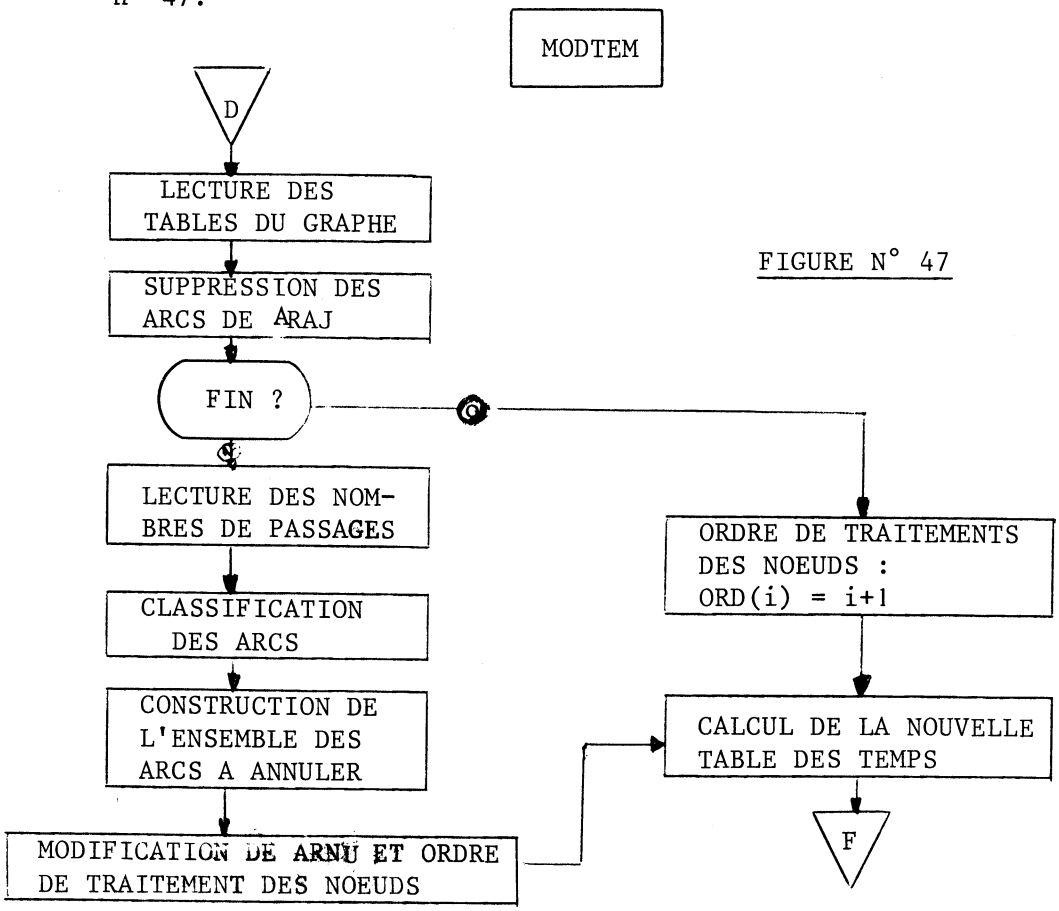


FIGURE N° 47

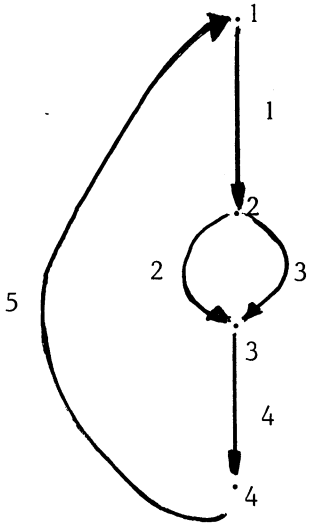
Nous pouvons remarquer que la modification de la table des temps se fait de la même façon dans le cas des deux minimisations à partir des tables ORD et ARNU qui contiennent respectivement l'ordre des noeuds pour l'application des transformations Tvi,ni de T et la liste des arcs dont nous allons annuler les valeurs.

Au départ, nous commençons par supprimer les arcs de ARAJ qui ne correspondent en fait à aucun élément de la table des temps. Soit d un arc de ARAJ qui a le noeud ni pour origine et le noeud nj pour extrémité. Pour supprimer d, nous remplaçons dans les tables NOD et NOF tout occurrence de ni par nj. Le noeud ni n'appartient alors plus au graphe.

Nous devons également modifier la table ARNU ainsi : si ARNU(nj) = d, nous mettons dans ARNU(nj) ce qui se trouve dans ARNU(ni).

EXEMPLE :

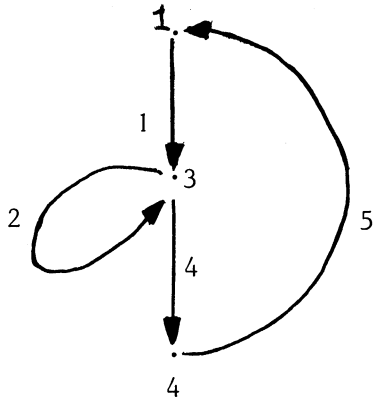
Soit le graphe de la figure n° 48. Il peut être représenté par les tables NOD, NOF, ARNU et ARAJ suivantes :



NOD	NOF	ARAJ	ARNU
1	2	3	0
2	3		1
2	3		3
3	4		4
4	1		

FIGURE N° 48

En supprimant l'arc 3, nous obtenons :



NOD	NOF	ARNU
1	3	0
3	3	1
0	0	1
3	4	4
4	1	

FIGURE N° 49

REMARQUES :

Nous aurions pu enlever les arcs de ARAJ au moment de la construction du graphe, mais la recherche de tous les arcs ayant pour extrémité ou origine le noeud qui se trouve ainsi supprimé est beaucoup plus rapide à faire en LP70 qu'en METRA5.

IV.3.1. Cas de la minimisation en nombre

Dans le cas de la minimisation en nombre, l'ordre de traitement des noeuds pour la modification de la table des temps, n'est autre que celui de leur numérotation, c'est-à-dire :

ORD(i) = i+1 (nous commençons par le noeud 2)

La table ARNU des arcs dont nous allons annuler les valeurs, quant à elle, reste inchanger puisque lors de la construction du graphe nous l'avons rempli en vue de cette minimisation en nombre des appels.

IV.3.2. Cas de la minimisation en moyenne

IV.3.2.1. Classification des arcs

Pour classifier les arcs suivant l'ordre décroissant de leurs nombres moyens de passage, nous allons utiliser deux tableaux P1 et P2, en simulant pour chacun d'eux le fonctionnement d'une pile.

Au départ P1 est initialisé avec la valeur 0 et P2 avec N-1, N étant le nombre d'arcs du graphe. Soit NMPAS, le tableau qui contient les valeurs des nombres moyens de passages des arcs.

Nous faisons : NMPAS(0) = 0, NMPAS(N+1) = ∞

L'infini représente en fait un nombre très grand par rapport aux valeurs normales des nombres de passages.

Au $i^{ème}$ pas, nous vidons P1 dans P2 où inversement jusqu'à ce que le nombre moyen de passage de l'arc numéroté i que nous voulons rajouter, soit compris entre ceux des éléments au sommet de P1 et P2.

Nous plaçons alors l'arc en question au sommet de P1.

EXEMPLE :

NMPAS :

0
10
2
5
20
3
∞

au départ P1 = (0)
P2 = (6)

Pas 1 :

NMPAS(1) = 10

NMPAS(0) $<$ NMPAS(1) $<$ NMPAS(6)

nous pouvons mettre l'arc 1 sur P1

P1 = (0, 1) P2 = (6)

Pas 2 :

NMPAS(2) = 2

NMPAS(2) $<$ NMPAS(1) on doit vider P1 dans P2

P1 = (0) P2 = (6,1)

NMPAS(0) $<$ NMPAS(2) $<$ NMPAS(1)

nous pouvons mettre l'arc 2 sur P1

P1 = (0,2) P2 = (6,1)

Pas 3 :

NMPAS(3) = 5

NMPAS(2) < NMPAS(3) < NMPAS(1)

P1 = (0, 2, 3) P2 = (6, 1)

Pas 4 :

NMPAS(4) = 20

NMPAS(1) < NMPAS(4)

P1 = (0,2,3,1) P2 = (6)

NMPAS(1) < NMPAS(4) < NMPAS(6)

P1 =(0,2,3,1,4) P2 = (6)

Pas 5 :

NMPAS(5) = 3

NMPAS(5) < NMPAS(4)

P1 = (0,2,3,1) P2 = (6,4)

NMPAS(5) < NMPAS(1)

P1 = (0,2,3) P2 = (6,4,1)

NMPAS(5) < NMPAS(3)

P1 = (0,2) P2 = (6,4,1,3)

NMPAS(2) < NMPAS(5) < NMPAS(3)

P1 = (0,2,5) P2 = (6,4,1,3)

Il ne reste alors qu'à vider P1 dans P2

P2 = (6,4,1,3,5,2,0)

Nous obtenons la classification : 4, 1, 3, 5, 2

IV.3.2.2. Construction de D l'ensemble des arcs à annuler

Nous appliquons purement et simplement l'algorithme A2 (paragraphe III.7.).

Soient d1 d2 ... dp les arcs du graphe ordonné par valeur décroissante des nombres moyens de passages. Pour voir au pas i si Dudi

TABARB

N° de noeud	N° d'arbre

contient un cycle ou non, nous construisons au fûr et à mesure une table TABARB représentant le graphe G'D formé dans le graphe réduit par les arcs de D. Nous y notons les noeuds adjacents à ces arcs et pour chacun d'eux l'arbre auquel il appartient.
(schéma ci-contre).

Les arbres sont numérotés par ordre d'apparition. Lorsqu'un arc, réunissant deux noeuds appartenant à deux arbres différents est rajouté, nous supprimons l'un de ces deux arbres qui ne font plus qu'un, en remplaçant partout dans la table TABARB, le numéro du premier par celui du second.

Pour voir si di constitue un cycle avec les arcs de D il suffit alors de voir si les deux noeuds qui lui sont adjacents appartiennent au même arbre ou non. Celà revient à parcourir une fois séquentiellement la table TABARB.

Cette méthode est beaucoup plus rapide que les applications successives d'un des algorithmes de détermination des cycles.

L'organigramme de la figure n° 50 explique comment se fait la construction de D, l'ensemble des arcs à annuler.

IV.3.2.3. Modification de ARNU et définition de l'ordre de traitement des noeuds

Nous appliquons l'itération I1 conformément à l'organigramme de la figure n° 51.

Nous utilisons une pile notée P.

Soit D =(d'1 d'2... d'n) l'ensemble des arcs à annuler.

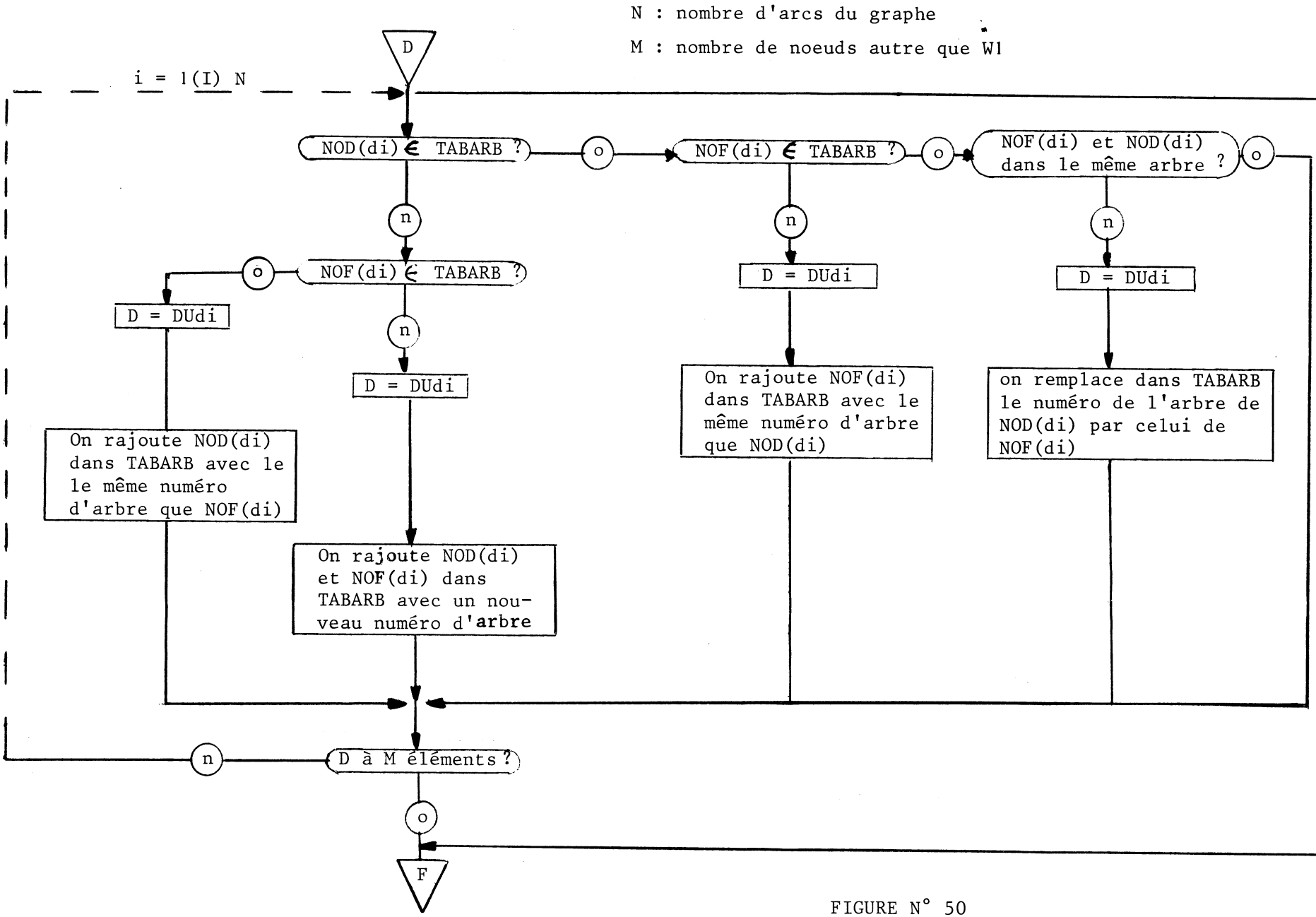

N : nombre d'arcs du graphe
M : nombre de noeuds autre que W1
D
i = 1(I) N
NOD(di) ∈ TABARB ?
o
NOF(di) ∈ TABARB ?
o
NOF(di) et NOD(di) dans le même arbre ?
o
n
n
n
o
NOF(di) ∈ TABARB ?
D = DUdi
D = DUdi
D = DUdi
n
On rajoute NOF(di) dans TABARB avec le même numéro d'arbre que NOD(di)
on remplace dans TABARB le numéro de l'arbre de NOD(di) par celui de NOF(di)
On rajoute NOD(di) dans TABARB avec le le même numéro d'arbre que NOF(di)
D = DUdi
On rajoute NOD(di) et NOF(di) dans TABARB avec un nouveau numéro d'arbre
n
D à M éléments ?
o
F

FIGURE N° 50

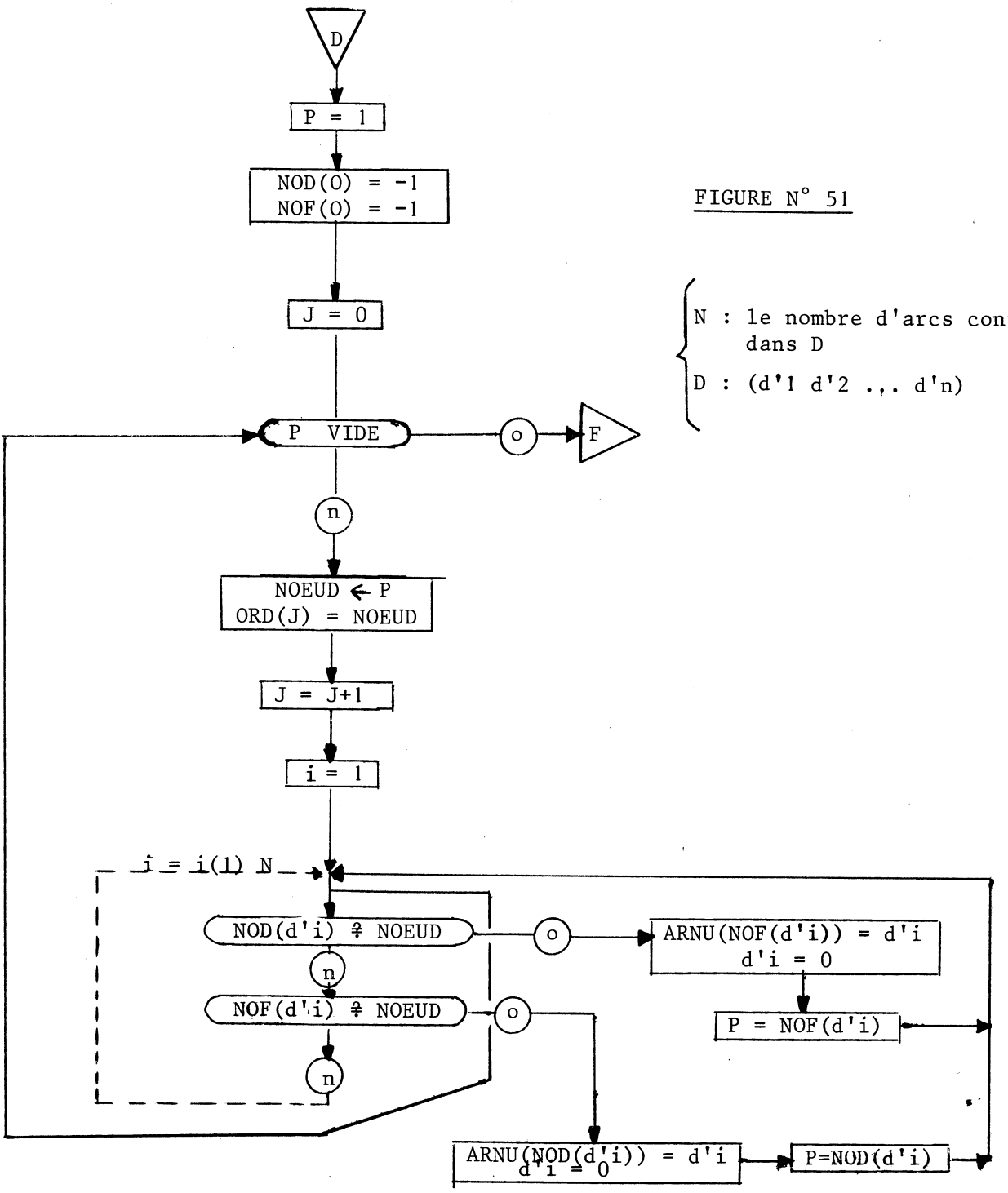


FIGURE N° 51

N : le nombre d'arcs contenus dans D

D : (d'1 d'2 .,. d'n)

Au départ, le noeud 1 (W1) est mis dans P. Chaque fois qu'un arc est mis dans ARNU, nous le supprimons de la liste D en les remplaçant par le numéro 0. NOD(0) et NOF(0) sont préalablement initialiser à la valeur - 1, l'arc 0 n'existant pas.

L'ordre de traitement des noeuds est alors fourni par leur ordre de passage dans la variable NOEUD.

IV.3.3. Modification de la table des temps

Elle se fait en deux temps à partir des tables ARNU et ORD :

-- Nous calculons successivement les valeurs vk des applications Tv2,n2 ... Tvk,nk, ... qui seront utilisées pour annuler les valeurs associées aux arcs de ARNU en suivant l'ordre sur les noeuds fourni par ORD. Ces valeurs sont stockées dans le tableau noté V.

-- Nous considérons que sur n1, qui est en fait W1, nous appliquons TV1,n1 avec V1 égale à 0.

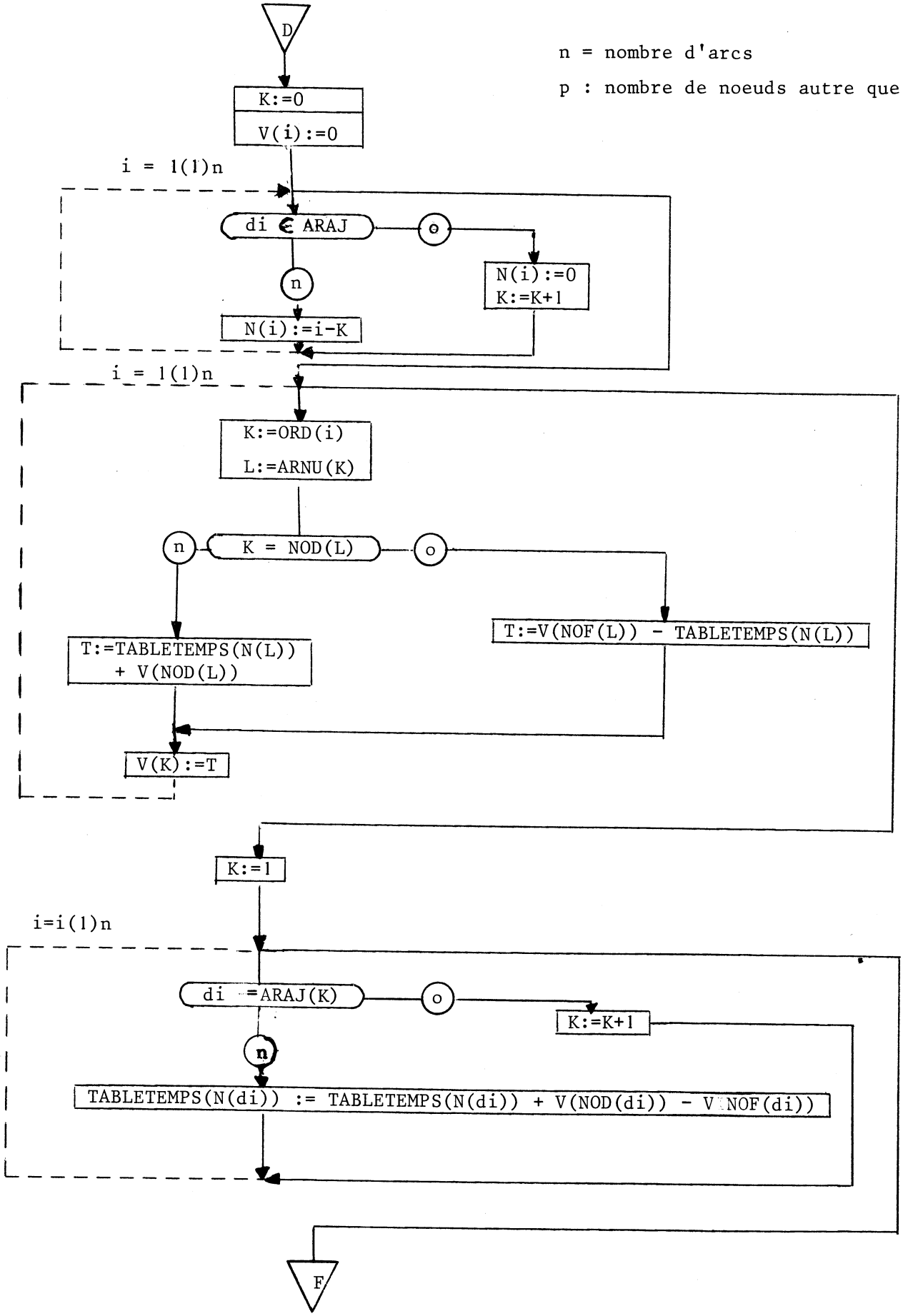
Alors, pour tout arc dp du graphe ayant ni comme origine et nj comme extrémité, la nouvelle valeur Fp de la table des temps sera :

$$Fp = tp + vi - vj$$

vi et vj étant les valeurs choisies pour les transformations Tvi,ni et Tvj,nj.

Un tableau noté N permet de faire correspondre aux arcs, l'indice de l'élément qui leur appartient dans la table des temps.

La figure n°52 représente l'organigramme de cette dernière partie.


n = nombre d'arcs
p : nombre de noeuds autre que W1
D
K:=0
V(i):=0
i = 1(1)n
di ∈ ARAJ
o
n
N(i):=0
K:=K+1
N(i):=i-K
i = 1(1)n
K:=ORD(i)
L:=ARNU(K)
n
K = NOD(L)
o
T:=TABLETEMPS(N(L))
+ V(NOD(L))
T:=V(NOF(L)) - TABLETEMPS(N(L))
V(K):=T
K:=1
i=i(1)n
di =ARAJ(K)
o
K:=K+1
n
TABLETEMPS(N(di)) := TABLETEMPS(N(di)) + V(NOD(di)) - V NOF(di))
F

IV.4.Obtention des nombres moyens de passages

Nous allons exposer deux méthodes différentes pour associer à chaque arc son nombre moyen de passages à l'exécution.

IV.4.1. Méthode par le calcul

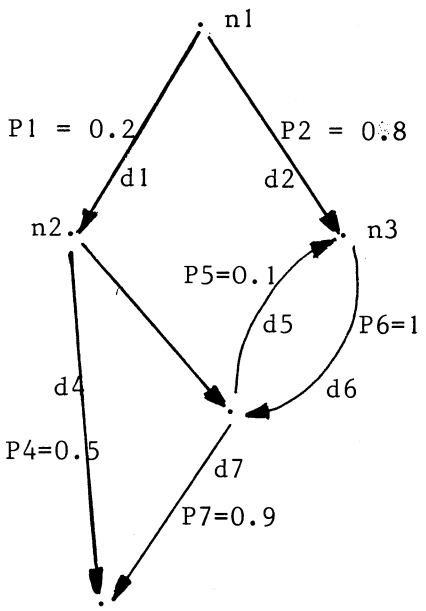


FIGURE N° 53

Etant donné un graphe G. Supposons que nous ayons pour chaque arc dp allant du noeud ni au noeud nj la probabilité Pp qu'il a d'être utilisé à partir de ni.

En partant de ces valeurs, nous pouvons obtenir par le calcul les nombres moyens de passages.

En effet, pour un arc dp :

$$np = \sum_{ck \in \mathcal{C}p} P(ck) \cdot NP(ck)$$

avec : $\mathcal{C}p(c1 \ldots ck \ldots)$, les chemins de la source au puits passant par l'arc dp.

P(ck) la probabilité du chemin ck

NP(ck) le nombre d'occurrences de dp dans ck.

L'application de cette méthode risque d'être longue. Nous pouvons cependant remarquer que pour les noeuds qui ne correspondent pas à un appel de procédure, les nombres moyens de passages des arcs adjacents à celui-ci, vérifient la loi de Kirchoff.

Cela permet de ne faire le calcul que pour un nombre minimum d'arcs [K2]. Nous en déduisons tous les autres.

La difficulté majeure de cette méthode consiste à associer simplement par le calcul ou par évaluation préalable, à chacun des arcs, une probabilité.

IV.4.2. Méthode par la mesure

Nous avons vu comment le programme GRAPHE2 rajoute au code du segment traité les instructions nécessaires à la mesure des nombres de passages par les arcs au cours d'une exécution.

Il suffit alors de réaliser un certain nombre d'exécution avec un échantillonnage de données assez diversifié pour représenter les exécutions possibles. Avec les résultats obtenus, nous faisons alors des moyennes.

La principale difficulté réside dans le fait de définir un échantillonnage de données qui soit valable et sur quels critères faire ce choix.

Dans le cas du problème qui nous a été posé, les programmes que nous traitons sont ceux qui composent le système mesuré et ceux qui sont destinés à faire tourner ce dernier. Les données ne varient pas d'une exécution à l'autre du modèle.

Nous prendrons alors pour les nombres moyens de passages par les arcs, les nombres de passages obtenus au cours d'une exécution unique des programmes utilisateurs sur le système que nous voulons mesurer, après leur instrumentation par GRAPHE2.

Bien que cette exécution ne se déroulent pas exactement de la même façon que sur la machine virtuelle, les fréquences de passages par les arcs doivent être du même ordre, les travaux réalisés étant en fait les mêmes.

V - RESULTATS OBTENUS

Les programmes exposés précédemment ont fait au cours de leur mise au point, l'objet de nombreux tests qui permettent d'avoir une idée sur l'ordre de minimisation que nous obtenons.

Si nous pouvons difficilement tirer des conclusions en ce qui concerne la diminution du temps d'exécution apportée par la minimisation en nombre, puisqu'elle dépend essentiellement des lieux d'implantation des appels, il n'en est pas de même pour ce qui est de la minimisation en moyenne.

Nous allons donner comme exemple, les résultats que nous avons obtenus en traitant notre propre programme LP70 de modification de la table des temps. L'organigramme de la figure n° 54 explique comment se déroule cet essai.

La table des temps que nous fournissons pour la minimisation est la suivante :

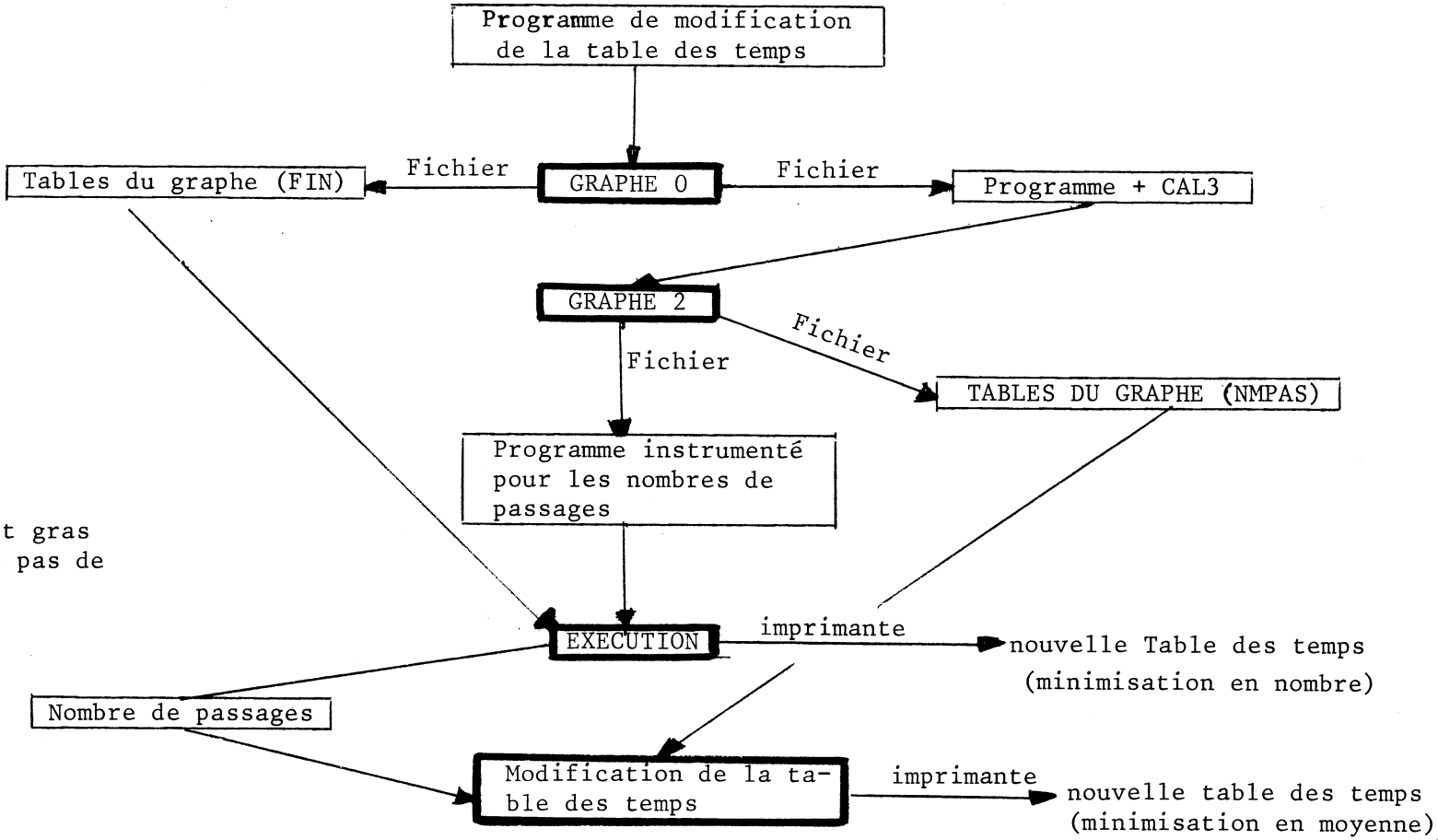
$$T(i) = i \qquad \text{pour } i = 1 \text{ à } N$$

N est le nombre d'arcs du graphe obtenu après analyse de MODTEM.

Le graphe de programme construit à partir de MODTEM comporte 145 arcs dont 12 appartiennent à ARAJ et 62 noeuds. Après suppression des arcs de ARAJ il reste 133 arcs et 50 noeuds.

La minimisation en nombre doit donc annuler 49 valeurs dans la table des temps.

Si nous rajoutons les 9 arcs ayant pour origine le noeud 0 qui sont en fait inaccessibles, nous devons obtenir 58 valeurs nulles dans la table des temps. C'est en effet le résultat que nous avons constaté. La solution correspond alors à une exécution moyenne de 8362 appels. Ce qui fait une réduction à 43,6 % des appels en nombre et 57,8 % en moyenne.



es cadres en trait gras
eprésentent les 4 pas de
exécution.

FIGURE N° 54

Pour la minimisation en moyenne, nous obtiendront le même nombre d'appels insérés dans le programme. A l'exécution, le nombre moyen d'appels sera 6393 au lieu de 14449 soit une réduction à 44,2 %.

Supposons que le modèle soit amené à simuler une dizaine de minutes le déroulement du système mesuré sur la machine virtuelle. En mettant un appel par branche, nous avons vu que le nombre d'instructions exécutées sur la machine virtuelle se trouve multiplié par 5, ce qui fait environ 50 minutes d'exécution.

En diminuant de moitié à peu près le nombre des appels, comme cela est souvent le cas pour la minimisation en nombre, le temps d'exécution du modèle est réduit à 35 minutes, alors que cette minimisation demande 2 minutes de plus à l'exécution que l'instrumentation de toutes les branches.

Pour la minimisation en moyenne, la mesure des nombres de passages demande une vingtaine de minutes environ. Pour environ 24 minutes de traitement, nous reduisons à 23 Minutes le temps d'exécution. Ce qui est beaucoup plus intéressant si nous voulons utiliser plusieurs fois le modèle instrumenté en modifiant par exemple certains composants de la machine virtuelle.

VI. CONCLUSION

Le modèle de simulation que nous avons présenté a une application restreinte à des systèmes relativement simples et écrits dans un langage facilement analysable comme LP70.

La détermination des points de synchronisation est pratiquement impossible à réaliser sur un gros système et la construction du graphe est difficile à faire pour des programmes écrits en assembleur où les adresses de branchement peuvent être calculer arithmétiquement et ne correspondent pas forcément à une étiquette.

Il est alors plus aisé de faire un chronométrage du déroulement de l'unité centrale. Les temps d'exécution des instructions ne sont plus alors des temps simulés mais ceux de la machine réelle sur laquelle se trouve le modèle. Les seules variables de la machine virtuelle sont alors la nature et le fonctionnement des périphériques et la taille de la mémoire centrale.

L'application que nous avons traitée ne semble pas avoir en elle-même une grande portée générale. Cependant, le but et l'intérêt principal de cette thèse est de montrer sur un exemple comment en représentant des programmes par des graphes, nous pouvons résoudre des problèmes de programmation qui sans cela semblent difficilement abordables.

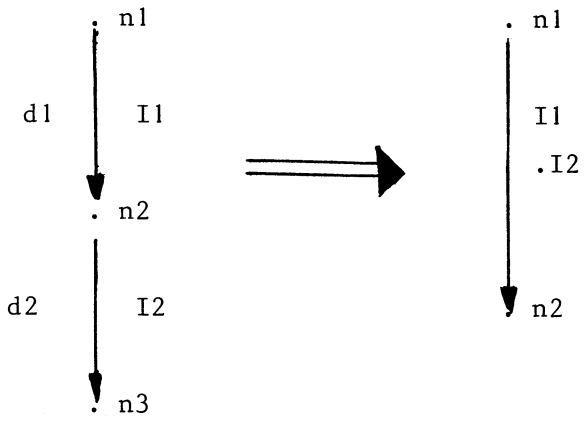
En outre, elle permet de voir comment par une analyse syntaxique, nous pouvons construire pour chaque programme, le graphe lui correspondant. Cette construction est à la base de toute utilisation de ces modèles. C'est une partie importante de ce travail.

Il existe une grande quantité d'applications possibles de ces graphes. Certaines ont déjà fait l'objet d'études et de publications comme par exemple les problèmes de segmentation de programme [L1]. Ils se ramènent en fait, à une étude de la distribution des boucles : on évitera en effet de mettre à cheval sur deux segments une boucle utilisée un grand nombre de fois.

Nous pouvons également, nous en servir pour améliorer la structure d'un programme, au moyen de transformations systématiques. Ces transformations peuvent éventuellement être décrites sous forme de règles.

Soit par exemple la règle :

(R1)



. avec la restriction que n2 est un noeud ayant aucun arc qui boucle sur lui.

I1 et I2 sont respectivement les suites d'instructions représentées par d1 et d2.
I1 . I2 est la concaténation de I1 et I2.

Nous allons appliquer cette règle, autant de fois qu'il est possible, sur le graphe de la figure n° 55

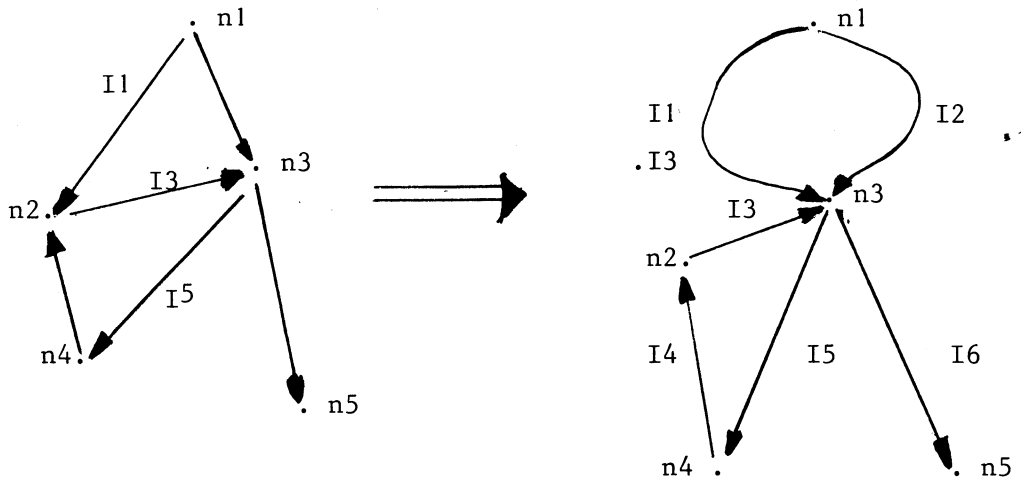
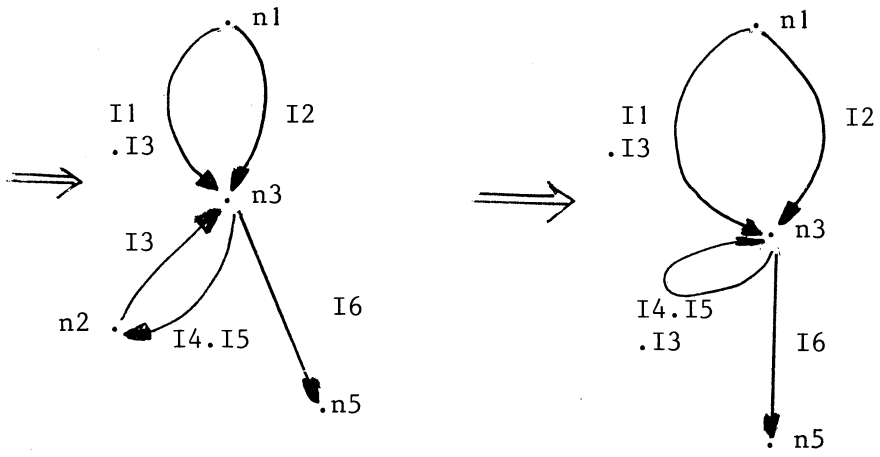


FIGURE N° 55



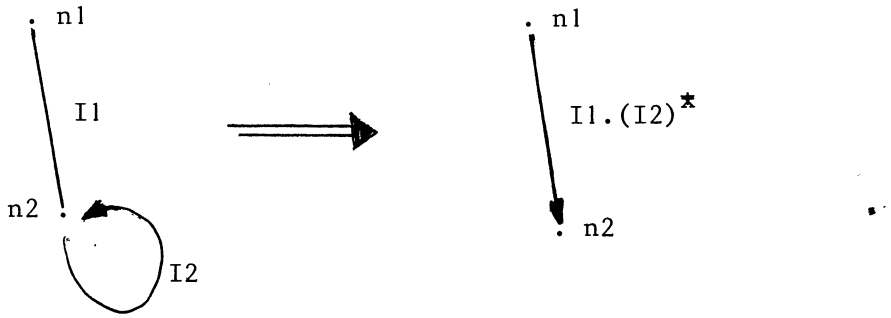
Nous avons ainsi obtenu un graphe équivalent à celui dont nous sommes partis et qui possède 4 arcs au lieu de 6.

REMARQUES :

-- L'application successive de la règle R1, réalise une réduction du graphe proche de celle de GRAHAM [G1]. Seul les boucles et les noeuds qui leurs sont adjacents, demeurent.

-- Nous pouvons considérer le graphe que nous traitons comme un automate d'état fini avec le vocabulaire I1, I2, ... In.

Soit R2 la règle suivante :



L'application successive des règles R1 et R2 sur le graphe n'est autre que l'algorithme de construction du langage régulier y correspondant [V1]. Pour le graphe de la figure n° 55 nous obtenons :

$$I1.I3 + I2 \,.\, (I3.I4.I5)^{*}\ I6$$

KNUTH et FLOYD se sont servis des grammaires de schémas pour démontrer que tout branchement au moyen d'une instruction GOTO dans un programme peut être supprimé; à la fin il ne reste plus que des boucles et des instructions conditionnelles.

Nous avons vu précédemment comment nous pouvions obtenir par réduction du graphe le temps moyen d'exécution du programme qui lui est associée [G1] . Nous pouvons envisager au moment de la compilation d'un programme de construire le graphe qui lui correspond et de faire une évaluation préalable de ses performances. Nous pouvons également fournir les équations de KNUTH donnant les inconnus qui caractérisent une exécution particulière. Une autre utilisation possible de ces graphes consiste en la simulation du fonctionnement des programmes qu'ils représentent. Ce qui peut être fait dans un but de mise au point.

Nous pourrions certainement imaginer encore de nombreuses applications.

Tous ces problèmes ont été en fait ramenés dans le contexte de la théorie des graphes et peuvent posés, sous cette forme, s'avoisiner à d'autres provenant de domaines totalement différents.

Par exemple, l'algorithme de réduction successive que nous avons vu précédemment est à peu de chose près l'algorithme de réduction des graphes de transfert [B4].

Bien que les graphes de programme font pour l'instant davantage partie de la théorie que de la pratique, je pense qu'ils offrent un outil facile à manipuler pour représenter ce qui se passe en machine.

Il feront certainement encore l'objet de nombreuses études et de nombreuses applications.

B I B L I O G R A P H I E

[B1] C. BERGE "La théorie des graphes et ses applications"
DUNOD, PARIS 1958

[B2] J. BAER, D. BOVET, G. ESTRIN "Legality and other properties of graph models of computation"
J.A.C.M. vol. 17, n° 3, (july 70) pp. 543-554

[B3] J.C. BOUSSARD "Cours et travaux dirigés d'analyse syntaxique"
Grenoble 1969

[B4] M.M. BRZOZOWSKI et Mc CLUSEY " Signal flow graph techniques for sequential circuit"
I.E.E. Trans. Elect. Comp. 1963

[C1] COOPER D.C. "Computer program and graph transformations".
CARNEGIE INS. OF TECHNOLOGY (1966)

[E1] G. ESTRIN, R. TURN "Automatic assignment of computations in a variable structure computer system"
I.E.E.E. Trans. on Elect. Computers, vol. EC12 pp. 755-773 (décembre 1963)

[G1] GRAHAM "Performance prediction"
Proceedings of an Advanced School on Software Engineering (Munich 1972)

[H1] F. HARARY, R. NORMAN, D. CARTWRIGHT "Introduction à la théorie des graphes orientés"
DUNOD, 1968

[I1] IANOV "On the equivalence and transformations of program schemes"
DOKLADY AN USSR A113, 1957, pp. 39-42 et Trans Comm. ACM, vol. 1, n° 10, pp. 8-12 (octobre 1968)

[K1] J.R. KARP et R. MILLER "Properties of a model for parrallel computations : determinacy, termination, queuing"
S.I.A.M. J. Appl. MATH., vol. 14, n° 6, pp. 1390-1411 (novembre 1966)

[K2] D. KNUTH "The art of computer programming"
Vol. 1, "Fundamental Algorithms"
Addison-Wesley publishing compagny (1966)

[K3] D. KNUTH et FLOYD "Notes on avoiding "goto" statements"
Information processing letters (février 1971)
Vol. 1, n° 1, pp. 23-31

[K4] J. KUNTZMANN "Théorie des réseaux"
DUNOD, PARIS, 1972

[K5] J.B. KRUSKAL "On the shortest planny subtree of a graph"
Proc. AMS, 7, 1956, page 8

[K6] J.C. BOUSSARD "Cours et travaux dirigés d'analyse syntaxique"
Grenoble 1969

[L1] T. LOWE "Analysis of boolean program models for time-shared paged environments"
Communications of the ACM, vol. 12, n° 4,
April 1969

[L2] "Langage de programmation LP70"
CEGOS P38-0.50.T

[M1] D. MARTIN, G. ESTRIN "Models of computations and systems-evaluation of vertex probabilities in graph models of computations"
J. ACM, n° 14, pp. 287-299 (april 1967)

[M2] D. MARTIN, G. ESTRIN "Path lenght computations on graphs models of computations"
I.E.E. Trans On Elec. Comp., n° 18, pp. 530-536
(juin 1969)

[M3] D. MARTIN, G. ESTRIN "Models of computational systems - cyclic to acyclic graph transformations"
Trans. I.E.E., on Elect. Comp. n° 16, pp. 70-79
(février 1967)

[M4] "META5 Manual version for XDS Sigma 7 (or CII 10070) under batch processing monitor"
E. RUSSEL (IRIA)

[P1] PETRI C. "Kommunikation mit automaten"
Schiften des Reinisch - west - falichen inst.
instrumentell MATH., and der Univ. Bonn
Nr 2, BONN 1962

[R1] E. RUSSEL "Automatic assignment of computational tasks in a variable structure computer"
M.S. Thesis U. of California, Los Angeles, 1963

[S1] D. SLUTZ "The flow graph shemata model of parallel computation"
TH. MASS Inst. (septembre 1968)

[S2] "Project SAM : a kernel approach to system programming"
Third Internationnal symposium on computer and information sciences Miami (dec. 1969)

[S3] "10070 : moniteur de multiprogrammation SIRIS 7 - Manuel de présentation"
C.I.I. 3581 P1/FR

[V1] G. VEILLON "Application de l'algèbre des expressions régulières à à l'étude des cheminements dans les graphes finis"
Deuxième Thèse, Grenoble (avril 1970)

TABLE DES MATIERES