Un algorithme pour l'ordonnancement de tâches temps-réel sur des ressources non-préemptives

Alain Jorry

▶ **To cite this version:**

Alain Jorry. Un algorithme pour l'ordonnancement de tâches temps-réel sur des ressources non-préemptives. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1976. Français. NNT : . tel-00287122

HAL Id: tel-00287122

https://theses.hal.science/tel-00287122

Submitted on 11 Jun 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

l'Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR

par

Alain JORRY

UN ALGORITHME POUR L'ORDONNANCEMENT DE TACHES TEMPS-REEL SUR DES RESSOURCES NON-PREEMPTIVES

Thèse soutenue le 11 octobre 1976 devant la Commission d'Examen :

Président : Monsieur L. BOLLIET

Examinateurs : Messieurs R. BAUMANN
M. DEPEYROT
M. SAKAROVITCH
G. VEILLON

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : M. Philippe TRAYNARD

Vice-Président : M. Pierre-Jean LAURENT

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BLOCH Daniel	Physique du solide
	BONNETAIN Lucien	Chimie Minérale
	BONNIER Etienne	Electrochimie et Electrométallurgie
	BOUDOURIS Georges	Radioélectricité
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	COUMES André	Radioélectricité
	DURAND Francis	Métallurgie
	FELICI Noël	Electrostatique
	FOULARD Claude	Automatique
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean-Charles	Chimie-Physique
	PAUTHENET René	Physique du solide
	PERRET René	Servomécanismes
	POLOUJADOFF Michel	Electrotechnique
	SILBER Robert	Mécanique des Fluides

PROFESSEUR ASSOCIE

M.	ROUXEL Roland	Automatique

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuel	Electronique
	BOUVARD Maurice	Génie Mécanique
	COHEN Joseph	Electrotechnique
	LACOUME Jean-Louis	Géophysique
	LANCIA Roland	Electronique
	ROBERT François	Analyse numérique
	VEILLON Gérard	Informatique Fondamentale et Appliquée
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM.	ANCEAU François	Mathématiques Appliquées
	CHARTIER Germain	Electronique
	GUYOT Pierre	Chimie Minérale
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du solide
	MORET Roger	Electrotechnique Nucléaire
	PIERRARD Jean-Marie	Mécanique
	SABONNADIERE Jean-Claude	Informatique Fondamentale et Appliquée
Mme	SAUCIER Gabrièle	Informatique Fondamentale et Appliquée

MAITRE DE CONFERENCES ASSOCIE

M. LANDAU Ioan	Automatique

CHERCHEURS DU C.N.R.S. (Directeur et Maître de Recherche)

MM. FRUCHART Robert	Directeur de Recherche
ANSARA Ibrahim	Maître de Recherche
CARRE René	Maître de Recherche
DRIOLE Jean	Maître de Recherche
MATHIEU Jean-Claude	Maître de Recherche
MUNIER Jacques	Maître de Recherche

PREFACE

La soutenance de cette thèse représente, pour moi, à la fois un honneur et la marque d'une étape.

Un honneur, puisque les membres du jury ont accepté de se réunir pour juger mon travail.

Une étape, puisque les travaux menés dans le cadre du projet SPECTRE trouvent ici un aboutissement.

De cet honneur, je veux remercier

Monsieur le Professeur L. BOLLIET, qui, après avoir dirigé mes travaux depuis maintenant quatre ans, accepte aujourd'hui de présider le jury ;

Monsieur le Professeur R. BAUMANN, qui, après m'avoir permis de poursuivre et développer ces travaux dans le cadre de son équipe, est venu spécialement de MUNICH pour cette soutenance ;

Monsieur M. DEPEYROT, qui, de l'origine jusqu'à cet aboutissement m'a apporté aide et conseils, en tant que directeur du projet SPECTRE tout d'abord, puis, ce dont je lui suis profondément reconnaissant, après l'achèvement de ce projet, sur des bases uniquement personnelles ;

Monsieur M. SAKAROVITCH, qui a accepté de prendre le temps de juger ce travail ;

Monsieur le Professeur G. VEILLON, qui, après avoir été mon Professeur à l'ENSIMAG et m'avoir enseigné l'algorithmique, base de ce travail, a accepté de le juger ;

Pour cet aboutissement, je veux remercier

Ceux qui ont dirigé ces travaux, Monsieur le Professeur R. BAUMANN, Monsieur le Professeur L. BOLLIET, Monsieur M. DEPEYROT ;

Ceux qui, chercheurs, m'ont aidé de leurs conseils, ont guidé les premières démonstrations, ou ont aidé à la formulation, les membres du projet SPECTRE, et tout particulièrement Messieurs J-F. CHALLINE et F-J. RODRIGUEZ, les membres de l'équipe munichoise, spécialement Messieurs W. RÜB et G. SCHROTT.

Ceux qui m'ont fourni l'aide nécessaire pour les recherches bibliographiques, le personnel du centre de documentation de l'IRIA, particulièrement Mesdames A-F. BAUDRY, B. FURIC, P. TOUZEAU ;

Ceux qui ont réalisé ce document, les membres du centre de reprographie de l'USMG ;

Madame C. CHALAND, qui, depuis trois ans, m'a aidé à résoudre les problèmes administratifs posés par ce travail ;

Ceux qui, parents ou amis, ont accepté la polarisation, les heures indues et tous les défauts du chercheur, défauts que je crois avoir assez bien personnifiés pendant trois ans.

Mademoiselle C. CHAZELAS a assuré la dactylographie de ce document, y compris le dessin des figures ; elle a assuré la coordination entre toutes les personnes intéressées à cette recherche.

Pour ce travail et pour la qualité de sa réalisation, pour la façon dont elle a permis une collaboration rendue difficile par l'éloignement, qu'elle soit ici très sincérement remerciée.

à Raymond et Adrien

" Ne crains pas la perfection : tu ne l'atteindras jamais ! "

Salvador DALI. (Dix règles pour celui qui veut être peintre.)

RESUME

Ce document est la synthèse des travaux menés pour la résolution d'un problème d'ordonnancement, celui posé par le système Temps-Réel SPECTRE. (Divers types de ressources, plusieurs ressources par type, relations de précédence, arrivées échelonnées, dates critiques...).

La méthode utilisée pour parvenir à la solution et les réflexions successives y sont décrites et analysées.

De plus, ce travail décrit l'algorithme solution depuis sa définition jusqu'à sa programmation, en passant par la démonstration de sa validité.

TABLE DES MATIERES

LISTE DES FIGURES ET TABLEAUX

PLAN DE CE TRAVAIL. NORMES DE REDACTION.

Ce travail pourrait porter le sous titre "Recherche d'une solution pour un problème d'ordonnancement". C'est en effet la synthèse d'une étude qui de la définition du problème à la programmation de l'algorithme a permis de résoudre un problème d'ordonnancement. Nous avons voulu en extraire une méthodologie, ce pour trois raisons, tout d'abord parce que l'ordonnancement est aujourd'hui une juxtaposition de cas particuliers, pour lesquels tous les jours des dizaines d'algorithmes ou heuristiques sont développés ou modifiés, que les publications en sont pleines, aussi, l'apport d'un nouvel algorithme n'est pas un événement en soi, il est plus important de savoir trouver celui existant le plus proche de son problème et de l'adapter. Ensuite, parce que après avoir résolu le problème qui a fait l'objet de cette étude, nous nous sommes trouvés face à un second presque identique, pour lequel nous ne voulions pas réinvestir tout ce qui l'avait déjà été. Enfin, parce qu'une première version de ce travail devait être présentée à des étudiants comme séance de clôture d'un séminaire sur l'ordonnancement [2]. Leur imposer un nouvel algorithme alors qu'aucun n'en avait le besoin immédiat eut été inutile, mieux valait leur fournir une méthode qui le jour venu les aiderait à résoudre leur problème, lequel selon toute probabilité serait hors du domaine de validité des quelques algorithmes étudiés. Bien sûr, nous n'avons pas voulu rester dans le domaine de l'abstrait, c'est pourquoi cette méthode est illustrée par l'étude qui en est découlée, chaque chapitre correspond à une phase de cette étude, et elle y est décrite, analysée, c'est pourquoi l'on trouve à la fois la définition de l'algorithme, la preuve de sa validité et le programme qui le réalise, ainsi que les phases de définition du problème, d'analyse, de recherche bibliographique. Quand ce travail a été rédigé pour la première fois, nous ne disposions que des travaux de la première étude, maintenant que dix-huit mois se sont écoulés, et que la seconde arrive à la phase de réalisation, il est intéressant de constater que chaque chapitre de ce travail pourrait être illustré non plus par des documents issus de la première étude, mais par ceux rédigés depuis pour la seconde. Seul le manque de place l'interdisant. (Un coefficient 6 devant être appliqué en raison de la différence de complexité).

Le plan de ce travail est donc celui d'une telle étude, définition du problème, classement dans le domaine de l'ordonnancement pour aider à une recherche bibliographique, définition et étude de l'espace des solutions, étude d'une méthode de résolution, définition d'un algorithme, démonstration de ses propriétés, implémentations. En conclusion, nous avons ajouté une ouverture sur une étude complémentaire en vue de formaliser et modéliser l'ordonnancement et la recherche de solutions. En effet, si savoir résoudre les problèmes est une bonne chose, savoir ce qu'ils sont en est une meilleure, car c'est la seule qui puisse permettre de passer le mur de la complexité qui fait que l'ordonnancement est par trop une juxtaposition d'algorithmes avant que d'être une science, malgré les efforts entrepris. C'est pourquoi, outre la résolution de problèmes pratiques, nous nous sommes intéressés à cette étude formelle, introduite à la fin de ce travail [.1].

Les normes de rédaction sont :

a) A chaque chapitre correspondant à une phase de l'étude, nous en avons fait une partie indépendante, avec sa propre table des matières, son propre numérotage, et ses propres références. Le lecteur peut ainsi s'arrêter où il le désire dans la lecture, en particulier sauter le chapitre de démonstrations ou, pour le familier de l'ordonnancement, ou celui qui ne recherche qu'un algorithme, débuter à ce niveau, et même directement au listing. Il demeure néanmoins que la philosophie de ce travail réside dans l'analyse des phases SUCCESSIVES de l'étude.

b) [...] est un renvoi bibliographique à la fin du chapitre. Soit par numéro, s'il s'agit d'un article particulier, soit par nom d'auteur, s'il s'agit d'un ensemble de travaux.

c) (chap.V),
(§.),
renvoi à un chapitre (respectivement),
un paragraphe de ce document.

d) Un terme nouveau est introduit par une phrase qui lui sert de définition et dans laquelle il est souligné.

La méthode formelle :

Définition : ...

n'est utilisée que pour les objets mathématiques.

e) Il y a quatre symboles d'égalité qui sont :

$\triangleq$, = , := , ≡

$\triangleq$ signifie égal ou identique par définition ;

= signifie égal, c'est une valeur logique au sens d'ALGOL-60 ;

:= est le symbole d'affectation au sens d'ALGOL-60 ,

≡ signifie identique.

f) Chaque définition d'un concept particulier, ou d'un algorithme, est suivie d'une illustration prenant en compte tous les cas de figure. Ce travail se situant tantôt dans le domaine de l'informatique, tantôt dans celui de la recherche Opérationnelle, ces illustrations sembleront selon les cas inutiles au spécialiste de l'un ou l'autre domaine qu'il veuille bien alors passer à la suite.

---

[1] JORRY A. "Etude de la représentation des problèmes d'ordonnancement sous forme tensorielle. Réflexions préliminaire Rapport Interne LABORIA-SPECTRE IE/53. (Janvier 1975).

[2] JORRY A. "ORDON. Un algorithme pour l'ordonnancement de tâches Temps-Réel sur des ressources non-préemptives". Versic française de l'exposé fait dans le cadre du : Seminar Prozeβrechner. TUM.SFB/49-E2. (Février 1975).

INTRODUCTION

INTRODUCTION

La recherche de solutions pour des problèmes d'ordonnancement a donné naissance à une multitude de méthodes, et surtout à une multitude de systèmes de notation, tant pour l'exposé du problème, que pour la description de la méthode, et tout particulièrement pour la définition des contraintes et du domaine de validité de l'algorithme, ce dernier point étant par ailleurs toujours le plus obscur.

L'ordonnancement est un domaine extrêmement vaste, qui recouvre des phénomènes aussi différents que l'établissement d'un emploi du temps, la coordination d'un chantier, le séquencement des programmes dans un calculateur ; scientifiquement, il est rattaché à la Recherche Opérationnelle, ou à l'Informatique, mais nécessite des outils mathématiques ou statistiques de très haut niveau.

Ces deux points, complexité intrinsèque de la description, multiplicité des domaines d'étude et d'application, font qu'il n'existe pas de classification véritablement établie (1), les problèmes considérés comme les plus importants ont amené une multitude de recherches et d'algorithmes, et chaque nouvel essai est catalogué selon sa position parmi les problèmes de référence. Néanmoins, un certain nombre de qualificatifs semblent être aujourd'hui unanimement reconnus pour aider à définir le domaine d'application, et quelques autres pour illustrer la méthode, ils sont néanmoins extrêmement difficiles à traduire dans le langage courant des utilisateurs (1).

---

(1) Une telle classification générale semble être irréalisable tant qu'un formalisme suffisant n'aura pas été développé ; en effet, dans une étude limitée aux systèmes informatiques temps-réel [1], il a été dénombré 50 caractéristiques possibles des phénomènes mis en jeu, et la traduction en clair de ces caractéristiques ne nécessite pas moins de 64 pages. Par exemple, le nombre de ressources préemptives est une caractéristique primordiale du système, mais cette caractéristique n'est que la traduction en termes d'ordonnancement de phénomènes informatiques divers : Système de Gestion Mémoire, Système de Gestion des Fichiers, Système d'E/S...

Tant qu'un tel effort de vocabulaire et d'exhaustivité sera nécessair toute classification aura des entrées trop nombreuses pour être utilisable.

On peut citer ici trois essais de classification :

α) Un essai de formalisation [2].

β) Une classification selon la complexité de la méthode [3].

γ) Une classification selon le domaine de définition [4].

Néanmoins, faute de bien comprendre les problèmes auxquels on se trouve confronté, on en est réduit à décrire minutieusement, (laborieusement trop souvent) le cas particulier étudié, exprimer au mieux la solution adopt en se rattachant autant que faire se peut aux problèmes déjà étudiés, à moins bien entendu, qu'il ne s'agisse de présenter la $n+1^{ème}$ version de la solutio du problème n° ... déjà examiné n fois.

Faute de bien comprendre les problèmes auxquels on se trouve confr té, en effet, si le concept de base est aisé à saisir :

> "Face à un magma d'éléments séparés en tâches et ressources, "allou les unes aux autres de façon que ça tourne le mieux possible" en vérifiant certaines propriétés",

la nature intime de ces éléments, de leurs relations nous reste encore incon Le vocabulaire utilisé : tâche, processus, ressource, processeur, précédence, contrainte, temps de commutation ; manque encore de définitions précises, et est de fait redéfini à chaque fois, une tâche est pour l'un un bloc monolithique à exécuter sur une ressource, pour l'autre sur plusieurs ressources en même temps, pour un troisième sur plusieurs ressources séquentiellement, pour un quatrième elle est découpable en tranches, pour un cinquième sa durée varie, pour un sixième elle est stochastique, pour le septième elle peut être périodique, ce qui fait que le huitième parlera de famille de processus.

Si l'on ajoute le fait que l'on peut symétriquement allouer une ressource à une tâche, ou placer une tâche sur une ressource, allouer ou planifier ; si l'on tient compte des tentatives d'unification des concepts (précédence-temps de commutation) on obtient un fatras dont il est bien difficile de savoir comment l'aborder.

Dans un effort de clarification, il a été associé à chacun de ces termes les mots : ensemble, application, relation, mais la structure de ces ensembles, les propriétés de ces applications et de ces relations nous sont encore inconnues. A ce sujet, il est bon de saluer une étude où pour la première fois à ma connaissance, les résultats ne reposent que sur la structure algébrique de l'espace des solutions et ses propriétés [5].

Face à ces difficultés, c'est dans cette direction de structuration, analyse, classification, de recherche d'une terminologie propre que ce travail a été orienté, et ce de deux façons différentes, d'une part l'utilisation des modèles existants pour trouver des solutions, d'autre part la recherche de modèles nouveaux.

L'utilisation des modèles existants a fait l'objet de [6,7] et est seule reprise dans ce travail ; partant d'un espace des solutions prédéfini et propre, on cherche à le réduire sur la base des propriétés du type de problèmes particulier traité, puis on définit une méthode qui d'un problème spécifié fasse passer à sa (ses, une de ses) solution(s).

A la suite des travaux effectués avec Messieurs DEPEYROT, CHALLINE, RODRIGUEZ, ou par ces mêmes personnes [5,8,9,10], le modèle utilisé à l'origine s'est révélé trop rudimentaire, trop pauvre algébriquement, c'est pourquoi il a été affiné, en particulier, l'introduction d'une troisième dimension a permis de rejoindre la théorie mathématique utilisée en [5], la théorie des files d'attente, celle naissante des processus, ainsi que de faire le lien avec les outils de synchronisation. Cette extension du modèle n'est pas reprise ici, mais présentée en [11] elle est la suite logique de ce travail.

Enfin, en annexe est donnée la version ALGOL-60 de l'algorithme résultant de l'étude.

REFERENCES

[1] JORRY A. "Questions pour la réalisation d'un système d'ordonnancem destiné au Temps-Réel". (Mai 1975).

[2] KOHLER W.H. STEIGLITZ K. "Characterization and theoritical comparison of branch and bound algorithms for permutation problems". Journal of the ACM. Vol.21. N° 1. pp.140-156. (January 1974).

[3] COFFMANN E.G. Travaux menés à l'IRIA sur la complexité. (1974).

[4] JORRY A. "Bilan des connaissances sur l'ordonnancement des processu sur des ressources". Note de travail. (Septembre 1973).

"Bibliographie commentée sur le problème du job-shop scheduling". Note de travail. (Novembre 1973).

[5] DEPEYROT M. "Approche barycentrique par transformation de Fourier d'u classe de problèmes de placement et de transport". C.R. Ac Sc. Paris. Tome 279. (2 décembre 1974). Série A. pp.823-8

[6] JORRY A. "Ordon : Un algorithme pour l'ordonnancement de tâches temps-réel sur des ressources non-préemptives. Introducti à Ordon".
Version Française : Rapport interne LABORIA-SPECTRE IE/54. (Janvier 1975).
Version Anglaise : "International Workshop on real-time programming". Budapest. (Mars 1974). Et Rapport interne LABORIA-SPECTRE IA/115.

[7] JORRY A. "Ordon : Ein Zuteilungsalgorithmus für Realzeitsysteme". Seminaire Prozeßrechner. Tech. Univ. München. (20/02/1975).

[8] RODRIGUEZ F.J. "A rational approach for modular programming".
"Processor description and module definition".
"About Monitor's semantics".
Communication personnelle de trois manuscrits. (Avril 197

[9] CHALLINE J.F. "Une approche de l'ordonnancement des échanges selon le problème du commis-voyageur". IRIA-LABORIA. Rapport N° 10
(Février 1975).

[10] DEPEYROT M. "Une généralisation de la notion d'automate et applicatio
Thèse soutenue le 24 Juin 1975 à l'USMG (Grenoble).

[11] JORRY A. "Etude de la représentation des problèmes d'ordonnancemen
sous forme tensorielle. Réflexions préliminaires". Rappor
Interne IE/53. (Septembre 1974-Janvier 1975).

Chapitre premier.

Quelques réflexions sur les concepts mis en œuvre.

TABLE DES MATIERES

I. 1. AVERTISSEMENT

Nous voulons dans ce chapitre réfléchir sur le sens donné aux termes employés, et sur les buts recherchés. Il ne s'agit donc pas, contrairement à ce que laisse supposer la forme, de définitions au sens strict du terme, mais plutôt d'un regroupement des diverses significations invoquées par différents auteurs lors de l'usage des concepts étudiés ici, ce, dans le but d'en extraire ce qu'il y a de commun, et de retenir ce qu'il y a d'important et d'intéressant dans les nuances apportées. Au chapitre III, le lecteur trouvera une approche semblable à celle-ci, mais rigoureuse cette fois ; avec des définitions mathématiquement étayées. Néanmoins, cette étude informelle nous a semblé nécessaire, à la fois à la compréhension du problème et à la justification de l'étude formelle.

II. 2. LES ELEMENTS DE BASE

Nous distinguerons trois éléments de base pour les problèmes d'ordonnancement :

α) les objets,

β) les tâches,

γ) les machines,

utilisant ici le langage courant, sans aucune volonté de sous entendre quoi qu ce soit. Nous verrons comment ce langage courant se traduit en "argot de métier selon le domaine à planifier , parfois par les mêmes mots, mais toujours avec des précisions sous-entendues.

Toujours en utilisant le langage courant, nous dirons que :

- Un objet est un produit fini ou semi-fini, fabriqué ou transformé, ou assemblé par l'unité de production (atelier, usine, centre de calcul, école qu'il s'agit de planifier.

- Une tâche est opération de fabrication, de transformation, d'assemblage d'un objet.

- Une machine est l'outil qui sert à effectuer cette opération de fabrication, de transformation, d'assemblage d'un objet qu'est une tâche.

I. 3. LES PROPRIETES DES ELEMENTS DE BASE

Sans vouloir entrer dans la définition fine des diverses propriétés, du moins pas pour l'instant, on doit néanmoins constater que chacun des éléments de base apporte une série de contraintes intrinsèques, plus ou moins indépendantes. Si l'on se réfère à [11], ces contraintes intrinsèques sont la sémantique des éléments et peuvent s'exprimer sous forme de grammaires.

On obtient ainsi trois grammaires, G(P) pour les machines, G(D) pour les objets, G(K) pour les tâches. Si G(P) est pratiquement indépendante, G(D) et G(K) sont duales l'une de l'autre, [1], car le but d'une tâche est de fabriquer des objets, et un objet ne peut être fabriqué que par une (suite de) tâche (s).

Il faut insister sur le fait que les notations G(D), G(K), G(P) sont issues de l'informatique, mais que si elles sont employées ici à la place d'autres plus naturelles à ce niveau, ce n'est que par souci d'homogénéité, et que cela n'a rien de restrictif, ces concepts étant valables dans tous les domaines.

I.4.LES DIVERS POINTS DE VUE

Nous avons remarqué que objets et tâches étaient difficilement diss ciables, en menant l'étude plus à fond, on constate que toute caractéristique de l'un a son pendant (son dual [1]) chez l'autre. Ceci implique que dans tou les domaines de la planification , tantôt l'un, tantôt l'autre sera ignoré, l choix de l'un des deux se faisant selon l'importance relative que leur accord conceptuellement le planificateur.

Ainsi, dans une usine, où l'important est l'objet, c'est lui qui se l'élément considéré ; à l'inverse, dans un centre de calcul les résultats n'o frent aucun intérêt, (ils ne sont dépouillés que par l'utilisateur), l'élémen de base considéré sera alors la tâche (que l'on nomme alors programme ou job) de même dans une école, le bachelier n'est pas un élément intéressant en soi (1°/ on n'est pas sûr que le lycéen considéré le deviendra, 2°/ s'il le devie: il disparaît) l'important, c'est la tâche de préparation au bac, les cours, c sont alors les éléments de base. Pour les grands travaux également, existe la double possibilité, la planification tâche-transition ou la planification objet-transition, dans le premier cas la tâche est une flèche entre deux état: "objet", dans le second, l'objet est une flèche indiquant la succession de de tâches.

Ce phénomène est la manifestation dans le domaine de l'ordonnancemer de phénomènes bien connus par ailleurs : automates état-sortie ou automates transition-sortie, remplacement des instructions "goto" par des variables d'état...

Si l'on considère une usine, on peut ainsi dualement définir la succession des phases de fabrication par le BOP (Bill of Processes) [12], ou les divers états d'un objet par le BOM (Bill of Material) [12]. Il est équivalent de dire que les fabrications de A, B, C précèdent l'assemblage D, ou que l'objet D est composé des sous-objets ABC. (Noter l'inversion du temps [1]).

I. 5. LES ELEMENTS DU SECOND ORDRE

Aux propriétés intrinsèques données par G(D), G(K), G(P), (et dont nous n'avons pas encore examiné le détail), il faut ajouter des contraintes externes, motivées exclusivement par l'environnement, deux sont en particulier intéressantes, le coût et les délais. Nous ne mentionnerons que pour mémoire les tentatives pour traduire toute contrainte externe (et parfois même toute contrainte intrinsèque) sous forme de coût, si de fait dans le système économique actuel, toute violation d'une contrainte est pénalisée financièrement, la fonction de coût est alors trop compliquée pour être prise en compte, ou moralement insoutenable (coût de la mort d'un ouvrier).

Le coût et les délais sont deux types de contraintes qui vont compliquer le problème posé en nous obligeant à rechercher un optimum dans l'espace des solutions, alors que satisfaire à G(D), G(K), G(P) est une chose souvent simple.

I. 6. ETUDE SUCCINTE DE L'ORDONNANCEMENT EN MILIEU INFORMATIQUE

I. 6. 1. Avertissement.

Comme base de cette étude, nous prendrons le travail effectué par V. DONZEAU-GOUGE sur la notion de processus [2], en particulier pour ce qui est des définitions et des propriétés ; les concepts y sont semblables à ceux plus connus d'HABERMANN ou FRASER [3,4], mais la rigueur dans leur introduction en fait un outil plus propice à cette étude. Le résumé des définitions que nous donnons à la fin de ce chapitre détruit cette rigueur, il n'est destiné qu'à faciliter une première lecture. Le cas échéant, le lecteur voudra bien se reporter au document original.

I. 6. 2. Notion de processus de contexte, d'allocation.

Les machines portent en informatique le nom de ressources, et sont classées par BELL et NEWELL [5] en divers types : P,M,T,L,S... dont pour l'instant nous ne retiendrons que deux : P et M. P pour processeur est un élément actif qui transforme des objets nommés ici data ou informations, M pour mémoire est un élément passif qui emmagasine les objets entre deux transformations. Toutes les machines informatiques ou ressources ont un nom unique dit nom réel. A chaque suite de tâches informatiques ou procédure est associée une liste de noms virtuels représentant des machines virtuelles nécessaires à l'exécution des tâches. Cette liste est variable au cours du temps. Le but du moniteur est d'associer à chaque nom virtuel un nom réel, on dit alors que la ressource réelle x est allouée à la tâche t sous le nom virtuel y.

L'ensemble variable liste de noms virtuels, relations, liste de noms réels, constitue un contexte. Les variations de ce contexte se font à l'aide de trois primitives, demande (insertion d'un nom virtuel dans la liste), allocatio (insertion d'une relation et d'un nom réel), restitution (suppression d'un nom virtuel , de la relation qui le lie à un nom réel, et de ce nom réel).

La succession des diverses valeurs du contexte est un phénomène lié à l'environnement tout autant qu'à la procédure, c'est pourquoi deux exécutions de la même procédure fournissent deux suites différentes. Une suite de valeurs, liée à une exécution est nommée processus.

I. 6. 3. Ruptures, ruptures synchrones, ruptures asynchrones.

Une rupture est un évènement perturbant la succession logique des procédures, ou des instructions d'une procédure.

Il peut s'agir d'une volonté délibérée du programmeur (attente sur évènement, arrêt définitif, lancement d'une autre procédure...) ou d'un élément extérieur, (interruption, évènement, "trap", début, arrêt, reprise, fin d'une autre procédure...).

Dans le premier cas, le programmeur peut dire où dans son programme la perturbation se produira, elle est synchrone, dans le second cas, la perturbation se produit aléatoirement à n'importe quel endroit du programme, elle est asynchrone. On utilise également le terme interruption pour rupture asynchrone, et le terme déroutement pour les ruptures synchrones. Néanmoins, par souci d'homogénéité, nous nous en tiendrons à la terminologie de [2], en particulier pour éviter toute confusion quant au terme interruption.

I. 6. 4. Allocation et mode d'accès. Notion de niveau.

Reprenons le rôle du moniteur, nous avons dit qu'il était d'allouer une ressource réelle de nom réel x à une tâche t sous le nom virtuel y. Ceci correspond au fait que le moniteur met à la disposition de la tâche t un mode d'accès à la ressource x, ce mode d'accès est un algorithme, qui permet à la tâche t de déclencher les actions de x s'il s'agit d'une ressource active (P), ou de modifier x s'il s'agit d'une ressource passive (M). Il est bien entendu que ces termes s'entendent au niveau près (caractéristique fondamentale de PMS), c'est-à-dire que l'algorithme s'exécute sur les sous-composants de x, et qu'une ressource passive M a un sous-ensemble actif (extracteur mémoire par exemple), tandis qu'une ressource active a aussi des éléments mémorisateurs.

Cette notion de niveau est fondamentale, non pas comme partout en informatique à cause du "Stepwise refinement", mais parce que l'on a trop tendance à travailler à divers niveaux à la fois, ce qui dans le domaine de l'ordonnancement crée des problèmes insolubles. Si l'on prend le cas des étreintes fatales comme exemple, il s'agit souvent d'une séparation mal contrôlée d'une ressource de niveau i en ressources de niveau i-1, demandées séparement en dépit de leur unité sémantique. Pour une étude plus fine des modes d'accès et de leurs caractéristiques, en particulier du problème des niveaux renvoyons le lecteur à [6,7].

I. 6. 5. Quelques propriétés intéressantes des éléments considérés.

I. 6. 5. 1. Avertissement.

Notre but n'est pas ici de traiter exhaustivement toutes les propriétés possibles ; comme nous l'avons mentionné dans l'introduction un tel travail est pratiquement irréalisable, et de plus sans objet. Le lecteur trouvera en [8] l'étude exhaustive d'un cas particulier (Minicalculateur Temps Réel) qui outre les retombées pratiques (réalisation effective du système) permet de mesurer la complexité du problème et de voir une méthode de résolution.

I. 6. 5. 2. Les processus.

La propriété principale des processus est leur durée, et plus exactement la précision avec laquelle cette durée est connue. Ce peut être une constante de la procédure associée, ce peut être une fonction ou une variable aléatoire de faible variance, ce peut être une variable aléatoire de forte variance.

I. 6. 5. 3. Les ressources.

Nous avons vu que les variations du contexte se faisaient sur rupture synchrone ou asynchrone. Outre les constantes du contexte que nous nommerons ressources permanentes, on peut distinguer deux types de ressources, celles dont la valeur est altérée par les restitutions asynchrones, et celles qui ne le sont pas. Le problème ne se pose pas pour les restitutions synchrones, l'utilisateur en étant responsable. Les ressources dont le contenu n'est pas altéré par les ruptures asynchrones seront dites préemptives, le moniteur pouvant en exiger la restitution sans risque, celles dont le contenu est altéré seront dites temporaires, elles sont allouées pour le temps souhaité par l'utilisateur On doit constater deux choses :

α) Les ressources préemptives sont de fait des ressources actives P, l'action terminée elles n'ont plus de contenu et peuvent être restituées, tandis que les ressources temporaires sont des ressources passives M, servant à mémoriser un état, leur contenu est indispensable au bon déroulement du processus.

β) Ici également, la notion de niveau intervient, seule la partie active des ressources P est préemptive, seule la partie passive des ressources M est temporaire. D'où une complexification des algorithmes de restitution pour la sauvegarde de la partie passive des ressources P, et la préemption de la partie active des ressources M. (Sauvegarde des registres, préemption du "bus" Mémoire). Cette notion de niveau est particulièrement visible dans les concepts de ressources virtuelles qu'ils portent ou non ce nom.

Exemples de mémoires virtuelles : - Mémoire virtuelle (préemption de la MC).
- Mémoire partitionnée (préemption du bus, du décodeur...).

Exemple de processeur virtuel : - Multiprogrammation (Sauvegarde des registres)

I. 6. 5. 4. Les coûts.

Tout d'abord, les coûts liés aux délais, ce peut être une constante, une fonction linéaire, une fonction de degré n, ou transcendante, ou une fonction en escalier. Ces fonctions sont croissantes par rapport au temps, ce sont des fonctions d'insatisfaction ; dans le cas contraire, on prend leur inverse. Une fonction de coût à deux marches (a,∞) correspond à un problème Temps-Réel vrai, une fonction de degré ≥ 1 à un Temps Réel "Soft", une constante à une fabrication normale.

Un coût temps réel est à la limite entre coût et sémantique, il peut être selon les cas intégré à G(K) ou G(D), ou intégré aux coûts.

Outre les coûts liés aux délais, il existe des coûts liés à G(P), tels les temps de mise en œuvre, le coût de non utilisation, le coût de surutilisation.

Ces diverses fonctions de coût sont de plus, avant optimisation complétées par des coefficients correcteurs, dans ce domaine, on constate qu'après avoir longtemps négligé le coût des machines, au profit des seuls délais (temps de traversée ou through put), on en vient de plus en plus à maximiser l'utilisation des ressources du moins dans les systèmes aujourd'hui en service, car l'apparition des microprocesseurs provoque un nouveau retournement de situation.

I. 7. TRANSCRIPTION DES RESULTATS

I. 7. 1. Introduction.

Si nous avons mené l'étude précédente dans le cadre de l'informatique, c'est d'une part parce qu'il nous est le plus familier, d'autre part et surtout parce qu'il est le seul à offrir les outils logiques nécessaires. Il faut cependant noter que les résultats en sont généraux, même s'ils ne disposent pas ailleurs du même support théorique.

I. 7. 2. Transcription.

Nous avons déjà noté l'existence de la dualité transition-état dans tous les domaines de l'ordonnancement, dualité qui se traduit par deux types de représentations graphiques, ou deux types de bordereaux (BOP, BOM), ce ne sont pas les seules constantes, les notions de stockage, de préemption, de temps de mise en œuvre demeurent ; un tracteur peut être dételé (préemption) un chariot ne peut être renversé pour le vider de son contenu (temporaire) ou il faut créer un autre stock (ressources virtuelles) ; les "bleus" de travail sont alloués en permanence à l'ouvrier. Les mêmes types de coût se retrouvent également fonctions en escalier pour les pénalités de retard, coût de mise en œuvre (changement des outils ou matrices) pour les machines, coût constant de la production pour le stock ; les diverses durées également, constante de la chaine, aléatoire de faible variance pour les rejets sur défaut, aléatoire de forte variance pour les grèves ou le mauvais temps (sauf au Sahara).

I. 8. L'ORDONNANCEMENT

I. 8. 1. Définition.

Nous dirons ainsi, qu'ordonnancer, ou planifier un ensemble de tâches sur un ensemble de machines, c'est définir pour chaque tâche la suite de machines, ou la suite d'ensembles de machines sur lesquelles elle s'exécutera ; et pour chaque machine la suite de tâches ou la suite d'ensembles de tâches qu'elle exécutera. Ou bien dualement pour chaque objet définir sa trajectoire à travers l'unité de production (ensemble des machines), et pour chaque machine sa trajectoire à travers l'espace des objets (*). Le tout, devant être intrinsèquement cohérent, cohérent avec G(D), G(K), G(P), et minimiser une combinaison des fonctions de coût choisie.

I. 8. 2. Propriétés.

Nous voulons distinguer divers modes d'ordonnancement, selon la façon dont se décide l'ordre des tâches, la fréquence de décision, le nombre des tâches considérées, là aussi une justification théorique est donnée en [14], nous nous limiterons donc ici à constater avec le vocabulaire usuel, une distinction habituelle, consacrée par l'usage avant que d'être justifiée par la théorie, si tant est que la théorie justifie les faits plus qu'elle n'est influencée par eux.

En allant vers la simplicité croissante de l'algorithme, (pas obligatoirement du problème), nous pouvons distinguer :

- l'ordonnancement statique,
- l'ordonnancement incrémentiel (ou incrémental),
- l'ordonnancement dynamique,
- la synchronisation.

(*) Cette image qui peut sembler découler d'une mauvaise assimilation de la relativité est de fait un phénomène réel, qu'il s'agisse de l'affectation des engins de chantier ou du problème du voyageur de commerce.

L'ordonnancement statique est réalisé par un algorithme qui considère un ensemble de tâches avant tout début d'exécution, calcule toutes les successions pour la période nécessaire à l'exécution de l'ensemble, est appelé pour chaque nouvel ensemble de tâches, et tel que les ensembles de tâches se succèdent dans l'unité de production de façon totalement indépendante.

L'ordonnancement incrémentiel est lui plus évolué, l'algorithme est capable, après avoir planifié un ensemble de tâches comme précédemment de teni compte de certaines modifications de cet ensemble (ajout ou retrait d'une tâche modification de certaines caractéristiques), sans avoir à reprendre tous les calculs, avec un coefficient de dégradation faible, (*) ou un coût de dégradation faible (*). Coefficient de dégradation faible pour les grands projets pour lesquels le temps de calcul du planning est négligeable, coût de dégradation faible pour les problèmes informatiques où travail et planification ont la même échelle de temps, se déroulant dans le même calculateur.

Si le coefficient de dégradation est nul, et si le coût de dégradation est nul ou négatif, la partie statique devient inutile ou nuisible, on utilise alors l'algorithme uniquement en mode incrémentiel, il est appelé à chaque modification et considère toujours l'ensemble des tâches, il est alors dynamique, il ne prévoit plus l'avenir jusqu'à la totale exécution de l'ensemble des tâches, mais uniquement jusqu'à l'instant de la prochaine modification.

(*) En posant : C1:: = coût du planning obtenu en faisant travailler l'algorithme en mode incrémentiel.
C2:: = coût du planning obtenu en faisant travailler l'algorithme en mode statique sur le nouvel ensemble (supposé optimal).
C'1:: = coût du calcul en mode incrémentiel.
C'2:: = coût du calcul en mode statique.

On a : Coefficient de dégradation : (C1-C2)/C2.
Coût de dégradation : ((C1+C'1) - (C2+C'2)).

Si de plus il n'est plus besoin de considérer l'ensemble des tâches, mais uniquement des sous-ensembles, s'il ne s'agit plus de prévoir la succession des tâches, mais simplement de régler au coup par coup les conflits possibles avec l'une quelconque des grammaires (G(D), G(K), G(P)), on obtient des problèmes de synchronisation. Exemple : G(P) indique qu'une machine a un degré de réentrance n. On contrôle l'accès par un sémaphore initialisé à n.

Là aussi, la notion de niveau intervient, une primitive de synchronisation masque un algorithme d'ordonnancement (même s'il s'agit d'un choix au hasard) ; les paquets d'échanges (ensemble d'ordres d'E/S dans SPECTRE)[13] sont ordonnancés par un algorithme (heuristique) statique [9], le sémaphore d'une page de mémoire virtuelle, cette simple primitive binaire,passe passe-pas , masque un algorithme de remplacement important, à l'inverse, l'algorithme statique le plus compliqué peut être traduit à l'exécution par un ensemble de file d'attente [10].

REFERENCES

[1] DEPEYROT M. "Une généralisation de la notion d'automate et applications Thèse soutenue le 24 Juin 1975 à l'USMG (Grenoble).

[2] DONZEAU-GOUGE V. "Etude de la gestion des contextes et de son application au temps-réel".
Thèse soutenue le 20 Juin 1974 à l'université de PARIS VI.

[3] HABERMANN A.N. "On the harmonious co-operation of abstract machines". Thès Technische Hogeschool Eindhoven. (October 1967).

[4] FRASER A.C. "On the meaning of names in programming systems". Comm. ACM Vol.14. N° 6. pp.409-416. (June 1971).

[5] BELL C.G.
NEWELL A. "Computer structures : readings and examples". Mc Graw Hill (1971).

[6] JORRY A. "Communication processus-ressources". (Août 1973).
"Allocation et modes d'accès". (Octobre 1973).
Notes de travail.

[7] CHALLINE J.F. "On the access mode in machine commands". (October 1973).
Notes de travail.

[8] JORRY A. "Questions pour la réalisation d'un système d'ordonnancemen destiné au Temps-Réel". (Mai 1975).

[9] CHALLINE J.F. "Une approche de l'ordonnancement des échanges selon le problème du commis-voyageur". IRIA-LABORIA. Rapport N° 105. (Février 1975).

[10] JORRY A. "Comment, dans le cadre du système Temps-Réel développé à l'Université Technique de Munich, réaliser un système d'ordonnancement déterministe". (Février 1975).

[11] DEPEYROT M. "Notes sur la conception structurée des interfaces. [Première partie]". IRIA-LABORIA. Rapport N° 131. (Juin 1975).

"Notes sur la conception structurée des interfaces. [Deuxième partie]". IRIA-LABORIA. Rapport N° 132. (Juin 1975).

[12] DEPEYROT M. "Le point sur nos discussions concernant la Robotique". (Juillet 1975).

[13] BANCILHON F. et al. "Principes de conception automatique dans le système SPECTR RAIRO. Vol.7. N° B-3. pp.31-62. (Octobre 1973).

[14] JORRY A. "Etude de la représentation des problèmes d'ordonnancement sous forme tensorielle. Réflexions préliminaires". Rapport Interne IE/53. (Septembre 1974-Janvier 1975).

ANNEXE : DEFINITIONS EXTRAITES DE [2]

Nous appellerons procédure une suite d'instructions portant sur des objets désignés par leur nom, chaque instruction étant exécutable sur un même processeur.

Intuitivement un corps de processus désigne la procédure et l'ensemble des informations qui sont nécessaires pour permettre et suivre son exécution et donc représenter la notion de processus.

Nous appellerons contexte un objet défini à partir de une ou plusieurs instructions et permettant d'associer à un ensemble de noms d'objets des informations suffisantes pour que les objets désignés soient directement accessibles par les instructions.

Une variable A_i est libre dans un contexte C si elle n'a pas dans C une valeur V_i associée.
Sinon A_i est dite liée.

Un contexte est entièrement spécifié si toutes ses variables sont liées.

Un contexte est associé à une suite non vide d'instructions. On peut donc associer un contexte à une procédure.

Mais on peut à partir d'une procédure définir toute une suite de configurations du contexte. Nous appellerons chaîne contextuelle C* cette suite.

Nous dirons qu'un objet est alloué relativement à un contexte si ce contexte peut prendre une configuration telle que le nom qui désigne l'objet est lié dans cette configuration. Allouer un objet à une suite d'instructions revient à insérer dans un contexte associé à cette suite d'instructions des informations suffisantes pour que l'objet soit directement accessible par les instructions de cette suite.

On appelle rupture tout élément perturbant le déroulement normal de l'exécution des instructions. Plus précisément nous avons supposé dans la définition de processeur que celui-ci, lorsqu'il exécute une instruction, calcule la position de l'instruction suivante qu'il va exécuter : si c'est cette instruction qui va effectivement être exécutée, nous disons que le déroulement de l'exécution est normal sinon une rupture se sera produite.

Une rupture peut être :

-une interruption externe provenant d'un calculateur analogique ou d'un processus industriel,

-une interruption provenant d'une horloge,

-une interruption provoquée par une fin d'exécution d'une autre instruction sur un autre processeur...

-un évènement.

Ces ruptures sont asynchrones par rapport au déroulement des instructions. Il existe également des ruptures synchrones qui sont :

-blocage sur un évènement ou une interruption,

- déroutement exemple : si valeur X = 0 alors fin

ou si valeur X = 0 alors

achever la procédure P_1.

Un corps de processus est une description de la machine abstraite construite à partir d'une procédure et qui va permettre d'introduire la notion de processus.

Nous appelons Corps de processus la machine abstraite définie à partir d'une procédure par le quintuple :

$\Sigma = (\mathcal{I}, \mathcal{E}, \mathcal{O}, \tau, \sigma)$ où

$\mathcal{I}$ est l'alphabet d'entrée de cette machine : $\mathcal{I}$ est l'ensemble des ruptures admissibles (cf.I.2.2.1) par la procédure.

$\mathcal{E}$ est l'ensemble des états du corps de processus. $\mathcal{E}$ est l'ensemble des configurations qui peuvent être créées à partir de la procédure ($\mathcal{E}$ est formé des mêmes éléments que la chaîne contextuelle C* définie à partir de la procédure) et que peut prendre le contexte associé à cette procédure. (Chaque élément de $\mathcal{E}$ représente un contexte défini à partir d'instructions de la procédure).

$\mathcal{O}$ est l'alphabet de sortie : c'est un ensemble de ruptures non forcément disjoint de $\mathcal{I}$.

τ est une fonction définie sur $\mathcal{E} \times \mathcal{I}$ à valeur dans $\mathcal{E}$ qui définit les transitions possibles sur $\mathcal{E}$.

σ est une fonction définie sur l'ensemble des éléments de $\mathcal{E}$ spécifiés et contenant la ressource privilégiée à valeur dans $\mathcal{O}$. Cette fonction définit le comportement externe du corps de processus.

Un processus est un quadruple (Σ, ei, W, VI) où :

- Σ est un corps de processus.
- ei est l'état initial du processus (point d'entrée) et ei appartient à l'ensemble $\mathcal{E}$ de Σ et plus précisément à la classe active.
- W est une séquence de ruptures admissibles par Σ. Par conséquent $W \in \mathcal{S}^*$
- VI est l'ensemble des valeurs que possèdent les objets dont les noms sont liés dans la configuration ei (en particulier VI spécifie les valeurs initiales des données du processus).

Chapitre second.

Une première classification des problèmes d'ordonnancement conduisant à un bilan des connaissances actuelles.

TABLE DES MATIERES

LISTE DES TABLEAUX

I. 1. AVERTISSEMENT

L'origine de cette classification est simple, elle mérite néanmoins d'être contée ; quand en août 1973, nous nous sommes intéressés au problème de planifier les processus tels que définis en [1], et possédant les propriétés exposées en [2], c'est-à-dire ceux du système Temps-Réel SPECTRE [2], notre premier travail fut d'examiner la littérature pour voir si un quelconque algorithme existait pour ce faire. Un premier critère de sélection des articles fut la durée des processus, nous n'avons considéré que les durées fixes. Un second critère fut le temps de mise en œuvre des machines, nous le voulions nul, nous sommes depuis revenus sur ce second critère pour [3]. Mais bien vite ces deux critères furent insuffisants, c'est ainsi que les propriétés énoncées au chapitre I furent peu à peu introduites et devinrent des critères. Ne voulant pas laisser perdre cet investissement bibliographique, nous avons collectionné et classé les articles, en même temps que faute de trouver un algorithme satisfaisant nous développions ORDON [4] et que l'étude des propriétés se complétait [5] et se formalisait [6,7]. De ce travail, il reste cette classification simple des problèmes à durée fixe, et une bibliographie commentée [8]. La première servant à ordonner la seconde, la seconde à illustrer la première.

La bibliographie commentée n'est pas reprise dans ce travail, par manque de place, si nous reprenons la classification, c'est qu'il nous semble qu'elle représente une phase importante du travail de recherche d'une solution. Après avoir examiné et défini son problème, (chapitre I), avant de déterminer l'espace des solutions (chapitre III), et d'en chercher une (chapitres III, IV, V), il est nécessaire de chercher à savoir s'il n'est pas résolu, et surtout (car il ne l'est jamais) quel est le coût d'une transformation qui le fera tomber dans le domaine du résolu.

C'est pourquoi, nous utilisons pour la classification un code binaire, (même si pour des besoins de compacité il est parfois écrit en décimal) ce type de codage, outre son extensibilité, (on peut ajouter autant de colonnes qu'on le désire) permet une définition simple du voisinage où l'on peut chercher un problème résolu, il suffit après avoir codé son problème, et vérifié qu'il

n'est pas résolu, de mettre une valeur indéterminée dans les colonnes de moindre importance, ou bien de regarder tous ceux qui ne diffèrent que par une ou i colonnes ou bien... Ce type de codage est également adapté à la factorisation (en algèbre de Boole) qui permet d'éliminer lignes et colonnes inutiles, ou de rédiger des questionnaires.

II. 2. LES CRITERES

II. 2. 1. Introduction.

Afin d'obtenir une représentation assez compacte pour figurer dans ce travail, nous avons limité le nombre de critères à six, dans un premier temps, nous utilisons la forme développée, donnant la réponse sous forme binaire. Puis après factorisation-élimination, la réponse est commentée, ces commentaires correspondant de fait à des critères complémentaires que nous n'avons pas voulu introduire dans la classification présentée ici. Ils le seraient si au lieu de ne classer ici que 64 cas possibles nous étions exhaustifs. Mais dans ce cas, le lecteur serait épuisé avant le sujet.

De ces 64 cas possibles, la factorisation fait passer à 16 dont 12 intéressants.

II. 2. 2. Critères déduits de G(P).

Ne voulant pas considérer le cas des ressources avec temps de mise en œuvre, nous conservons trois types de ressources :

- préemptives,
- temporaires,
- permanentes.

Un problème étant caractérisé par la présence ou l'absence simultanées ou non de ces types de ressource.

II. 2. 3. Critères déduits de G(K).

Deux critères ont été ici éliminés, la dépendance ou l'indépendance des tâches, et la connaissance de leur durée. Le premier est repris en critère secondaire dans la seconde classification. Pour le second, nous avons dit que nous considérerions des durées fixes. Pour les durées stochastiques, les méthodes sont par trop différentes, et les autres critères utilisés ici n'y ont pas leur place. C'est pourquoi, plutôt que de dédoubler ce travail, nous préférons renvoyer le lecteur à [9]. Néanmoins, pour les faibles variances, ou les valeurs

stochastiques bornées, les méthodes utilisées pour les durées fixes peuvent être valables, en particulier si le problème n'est pas résolu sous forme stochastique. (Un travail dans ce domaine est en cours qui fait suite à [5], et est destiné à prendre en compte les imprécisions).

Nous ne conserverons pour le premier temps qu'un seul critère, à mi-chemin entre G(K) et les coûts ; celui des échéances, pour classer les problèmes en deux catégories :

avec échéance critique (deadline)
sans échéance critique.

II. 2. 4. Critères déduits du type de la solution recherchée.

Des quatre types d'algorithmes, nous ne retiendrons que deux. Statiqu et dynamique. Les algorithmes incrémentaux n'ayant pas déchainé l'enthousiasme des scientifiques, ils restent à deux niveaux embryonnaires, les "bouche-trous" (on remplit les vides laissés par la phase statique) et le "savoir-faire" de contremaître. Ils semblent pourtant être la seule solution d'avenir aux problèmes de gestion de production.

Les algorithmes de synchronisation sont encore trop souvent des problèmes d'école, sauf s'il s'agit de synchronisation contrôlée, mais ils échappent alors à l'ordonnancement pour devenir des problèmes de graphes, résolus ou simulés (feux rouges et sens uniques...). Cependant, nous ajouterons un critère de mélange des niveaux, celui des étreintes fatales.

II. 3. LA CLASSIFICATION

La classification se présente comme un tableau à six entrées binaires et une sortie binaire (aux remarques près), les cas sont classés de 0 à 64 dans l'ordre naturel de la valeur des entrées.

Pour des raisons de mise en page, le tableau est scindé en quatre, et les remarques en sont extraites. Un chiffre entre parenthèses y renvoie le cas échéant.

Pour les cinq premiers critères, 0 correspond à l'absence, 1 à la présence de l'élément (de la propriété) considéré(e).

Pour le sixième critère, 0 correspond à un algorithme statique, 1 à un algorithme dynamique.

G(K)	G(P)			Solution		Réponse
Echéance critique	Permanentes	Temporaires	Préemptives	Etreintes fatales	Statique/ Dynamique	
0	0	0	0	0	0	(1) 0
1	0	0	0	0	0	(1) 1
0	1	0	0	0	0	Trivial (2) 2
1	1	0	0	0	0	Trivial (2) 3
0	0	1	0	0	0	Oui 4
1	0	1	0	0	0	Oui 5
0	1	1	0	0	0	Oui 6
1	1	1	0	0	0	Oui 7
0	0	0	1	0	0	Oui 8
1	0	0	1	0	0	Oui 9
0	1	0	1	0	0	Oui 10
1	1	0	1	0	0	Oui 11
0	0	1	1	0	0	Certains cas 12
1	0	1	1	0	0	Certains cas 13
0	1	1	1	0	0	Certains cas 14
1	1	1	1	0	0	Certains cas 15

Tableau 1 : Classification à six critères ($1^{ère}$ partie).

G(K)	G(P)			Solution		Réponse
Echéance critique	Permanentes	Temporaires	Préemptives	Etreintes fatales	Statique/ Dynamique	
0	0	0	0	1	0	(1) 16
1	0	0	0	1	0	(1) 17
0	1	0	0	1	0	Incohérent (3) 18
1	1	0	0	1	0	Incohérent (3) 19
0	0	1	0	1	0	Oui 20
1	0	1	0	1	0	Oui 21
0	1	1	0	1	0	Oui 22
1	1	1	0	1	0	Oui 23
0	0	0	1	1	0	Incohérent (3) 24
1	0	0	1	1	0	Incohérent (3) 25
0	1	0	1	1	0	Incohérent (3) 26
1	1	0	1	1	0	Incohérent (3) 27
0	0	1	1	1	0	Certains cas 28
1	0	1	1	1	0	Certains cas 29
0	1	1	1	1	0	Certains cas 30
1	1	1	1	1	0	Certains cas 31

Tableau 2 : Classification à six critères ($2^{ème}$ partie).

G(K)	G(P)			Solution		Réponse
Echéance critique	Permanentes	Temporaires	Préemptives	Etreintes fatales	Dynamique	
0	0	0	0	0	1	(1) 32
1	0	0	0	0	1	(1) 33
0	1	0	0	0	1	Trivial (2) 34
1	1	0	0	0	1	Trivial (2) 35
0	0	1	0	0	1	Oui 36
1	0	1	0	0	1	Non 37
0	1	1	0	0	1	Oui 38
1	1	1	0	0	1	Non 39
0	0	0	1	0	1	Oui 40
1	0	0	1	0	1	Certains cas 41
0	1	0	1	0	1	Oui 42
1	1	0	1	0	1	Certains cas 43
0	0	1	1	0	1	Oui 44
1	0	1	1	0	1	Non 45
0	1	1	1	0	1	Oui 46
1	1	1	1	0	1	Non 47

Tableau 3 : Classification à six critères ($3^{ème}$ partie).

G(K)	G(P)			Solution		Réponse
Echéance critique	Permanentes	Temporaires	Préemptives	Etreintes fatales	Dynamique	
0	0	0	0	1	1	(1) 48
1	0	0	0	1	1	(1) 49
0	1	0	0	1	1	Incohérent (3) 50
1	1	0	0	1	1	Incohérent (3) 51
0	0	1	0	1	1	Oui-non (4) 52
1	0	1	0	1	1	Non 53
0	1	1	0	1	1	Oui-non (4) 54
1	1	1	0	1	1	Non 55
0	0	0	1	1	1	Incohérent (3) 56
1	0	0	1	1	1	Incohérent (3) 57
0	1	0	1	1	1	Incohérent (3) 58
1	1	0	1	1	1	Incohérent (3) 59
0	0	1	1	1	1	Oui-non (4) 60
1	0	1	1	1	1	Non 61
0	1	1	1	1	1	Oui-non (4) 62
1	1	1	1	1	1	Non 63

Tableau 4 : Classification à six critères ($4^{ème}$ partie).

Remarques :

(1)	0,1,16,17,32,33,48,49	S'il n'y a pas de ressources, il n'y a pas problème.
(2)	2,3,34,35	La planification de ressources permanentes n'est pas un problème, sauf s'il s'agit d'u répartition.
(3)	18,19,24,25,26,27,50,51,56,57,58,59	Il n'y a étreinte fatale que pour des resso temporaires.
(4)	52,54,60,62	Une certaine connaissance à priori est néce

Tableau 5 : Classification à six critères (Remarques).

On constate, en regardant ce tableau, que la présence ou l'absence de ressources permanentes ne change rien à la réponse, en termes algébriques on note que $(Per + \overline{Per})$ est en facteur. On peut donc éliminer cette colonne.

On constate également que la prise en compte des étreintes fatales à ce niveau crée beaucoup de problèmes incohérents. Nous allons donc en faire un critère secondaire.

Ces deux remarques permettent de réduire le tableau de base à 16 cas, tandis qu'à l'inverse, les remarques s'allongent, on voit se dessiner l'arbre d'un questionnaire.

Numéro	Echéance	G(K) Temporaires	G(P) Préemptives	Dynamique	Solution
0	0	0	0	0	
1	1	0	0	0	
2	0	1	0	0	On sait de façon directe minimiser les fonctions de coût linéaires. On minimise en probabilité, ou avec des algorithmes combinatoires les autres fonctions.
3	1	1	0	0	Résolu par les méthodes combinatoires.
4	0	0	1	0	On sait minimiser directement beaucoup de fonctions de coût, tant que l'on ne tient pas compte du temps de préemption.
5	1	0	1	0	Le nombre de solutions directes est plus faible, il faut avoir recours à des heuristiques.
6	0	1	1	0	Semblable à 2 sous certaines conditions. On cherche à l'identifier à 2 ou à 4 par approximation, sinon on utilise des méthodes hautement combinatoires.
7	1	1	1	0	Idem 6, on cherche à l'identifier à 3 ou 5.
8	0	0	0	1	
9	1	0	0	1	
10	0	1	0	1	Sans étreintes fatales, on minimise en probabilité. Sinon, le dynamique vrai est impossible, des connaissances à priori sont nécessaires.
11	1	1	0	1	Non.
12	0	0	1	1	Beaucoup d'algorithmes minimisant toutes sortes de fonctions.
13	1	0	1	1	Si une seule ressource oui. Sinon non, on doit alors se ramener à 5.
14	0	1	1	1	Dans certains cas, identique à 10. Sinon on approxime via 10-12. Si étreintes fatales cf.10.
15	1	1	1	1	Non.

Tableau 6 : Classification réduite.

REFERENCES

[1] DONZEAU-GOUGE V. "Etude de la gestion des contextes et de son application au temps-réel". Thèse soutenue le 20 Juin 1974 à l'Université de Paris VI.

[2] BANCILHON F. et al. "Principes de conception automatique dans le système SPECTRE". RAIRO. Vol.7. N° B-3. pp.31-62. (Octobre 1973

[3] CHALLINE J.F. "Une approche de l'ordonnancement des échanges selon le problème du commis-voyageur". IRIA-LABORIA. Rapport N° 105. (Février 1975).

[4] JORRY A. "Ordon : A scheduling algorithm for real-time tasks on non-preemptive resources with deadline". IRIA-LABORIA. Rapport N° 64. (April 1974).

[5] JORRY A. Notes de travail :
"Comment, dans le cadre du système Temps-Réel développé à l'Université Technique de Munich réaliser un système d'ordonnancement déterministe". (Février 1975).

"Questions pour la réalisation d'un système d'ordonnancement destiné au Temps-Réel". (Mai 1975).

"Communication processus-ressources". (Août 1973).

"Allocation et modes d'accès". (Octobre 1973).

[6] JORRY A. "Ordon : Un algorithme pour l'ordonnancement de tâches temps-réel sur des ressources non-préemptives. Introduction à Ordon".
Version Française : Rapport Interne LABORIA-SPECTRE IE/54. (Janvier 1975).
Version Anglaise : "International Workshop on real-time programming". Budapest. (Mars 1974). Et Rapport Interne LABORIA-SPECTRE IA/115.

[7] JORRY A. "Etude de la représentation des problèmes d'ordonnai cement sous forme tensorielle. Réflexions prélimina: Rapport Interne LABORIA-SPECTRE IE/53. (Janvier 197!

[8] JORRY A. "Bibliographie commentée sur le problème du Job-Sho] scheduling". Note de travail. (Novembre 1973).

[9] LEROUDIER J. PARENT M. "Quelques aspects de la modélisation par simulation événements discrets". IRIA-LABORIA. Rapport N° 98. (Février 1975).

Chapitre troisième.

Une nouvelle approche des problèmes d'ordonnancement menant à l'analyse d'une méthode de recherche systématique et de son application à un problème donné : la recherche séquentielle en arbre avec limitation du nombre de nœuds considérés.

TABLE DES MATIERES

LISTE DES FIGURES ET TABLEAUX

III. 1. CONSIDERATIONS PRELIMINAIRES.

Nous cherchons à illustrer l'algorithme d'ordonnancement, par un ensemble de diagrammes d'occupation, ou Gantt-Charts.

La première figure représente l'exécution d'un ensemble de sept tâches, sur un ensemble de six ressources. Ce type de représentation toujours possible va nous aider à formuler le but recherché, et ainsi peut être, à l'atteindre.

Si l'on considère la tâche T1, on peut voir qu'elle est activée par trois fois, que ses ressources lui sont retirées après chaque activation, mais que ces mêmes trois ressources lui sont accordées à chaque fois. Ce que nous entendons par "activée par trois fois" c'est qu'elle a été activée, puis arrêté pour une quelconque raison, puis réactivée, non qu'elle a été activée par trois fois pour effectuer le même travail avec des paramètres différents. (i.e. ces trois activations relèvent du même processus).

Le but de tout algorithme d'ordonnancement, est de définir de tels diagrammes d'occupation pour l'utilisation future des ressources dont il a la charge, que cela soit fait explicitement (ordonnancement statique) ou implicitement par la formulation d'une règle de transition (ordonnancement dynamique Le terme utilisation future peut recouvrir une durée quelconque, qu'elle soit fixe, ou indéterminée (par exemple jusqu'à la prochaine interruption).

Dans la plupart des cas, il est inutile d'en arriver à une définition aussi précise, puisque l'on sait disposer d'un algorithme d'ordonnancement tel que tout diagramme généré satisfasse aux contraintes imposées. Dans le cas où l'on ne dispose pas d'un tel algorithme, il est alors nécessaire d'employer un algorithme de recherche systématique sur un certain espace de solutions possible Il n'existe pas de distinction fondamentale entre les deux approches, l'une pouvant être considérée comme un cas particulier de la seconde.

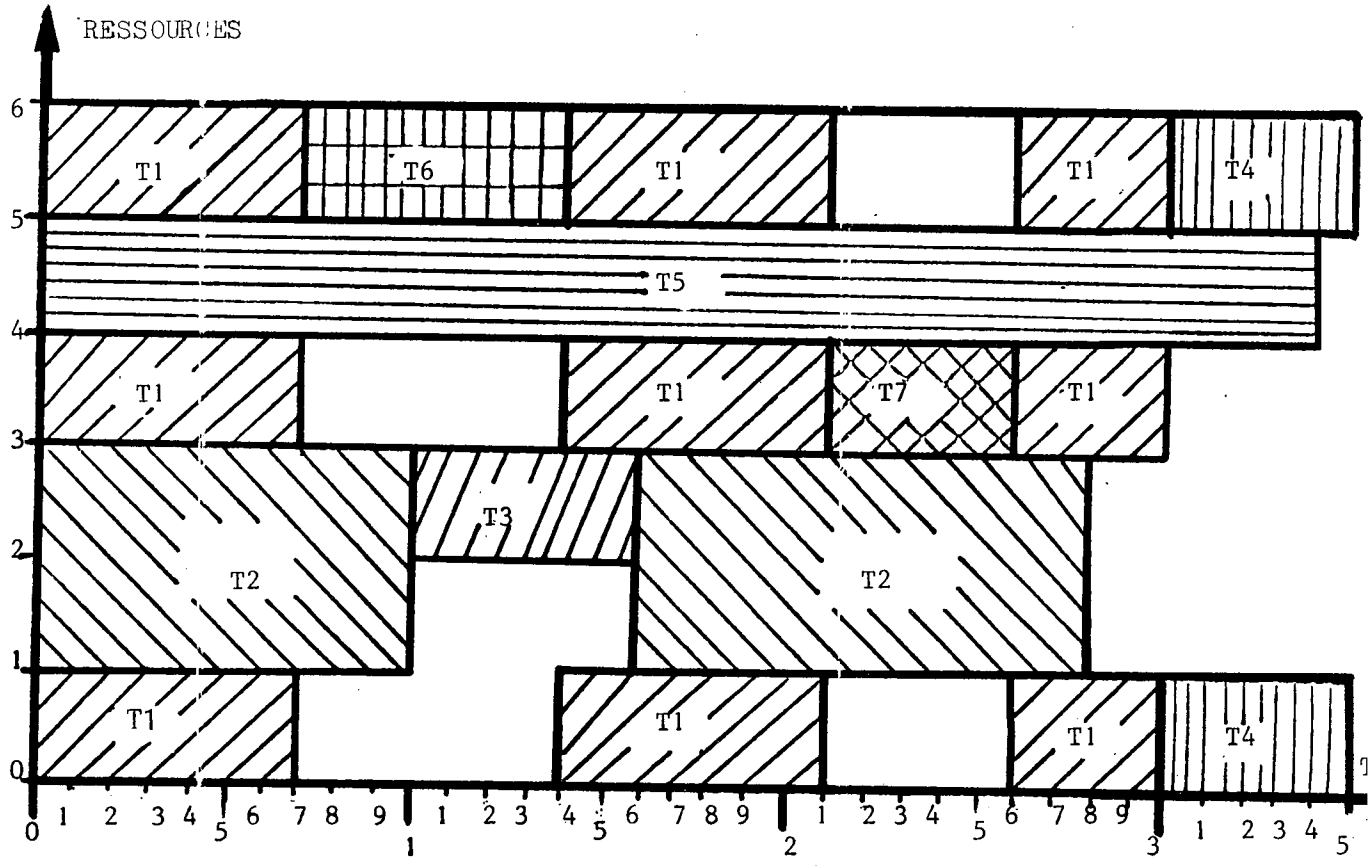


Figure 1 : Diagramme d'occupation de l'exemple.

III. 2. DEFINITION D'UNE RELATION ENTRE LES GANTT-CHARTS ET LES MATRICES.

Il est difficile de travailler les emplois du temps si on les laisse sous forme de diagrammes, c'est pourquoi notre premier objectif sera de définir une relation d'équivalence entre ces diagrammes, et des objets plus facilement manipulables. Nous avons choisi d'utiliser des matrices.

Ce peut être de simples copies des diagrammes, dans le cas de l'exemple une matrice 6×35 à valeurs dans $[0, 1, \ldots 7]$; ou tout autre type de matrices. Pour le cas que nous voulions traiter avec Ordon, nous avons choisi un autre type de matrices, mais la méthode utilisée pour limiter l'espace des solutions reste identique.

Considérons tout d'abord la partie du diagramme de l'exemple relative à la tâche numéro 1 (Fig. 2).

Puis introduisons un système d'identification afin de différencier les trois activations de cette tâche, par exemple numérotons les T1.1, T1.2, T1.3. (Fig. 3).

Essayons maintenant de mémoriser assez d'information sur ce diagramme pour être aptes à le reproduire. Un ensemble nécessaire et suffisant peut être :

- l'instant initial de chaque activation,
- la durée de chaque activation,
- la liste des ressources allouées pour chaque activation,

comme cela est fait dans la figure 4.

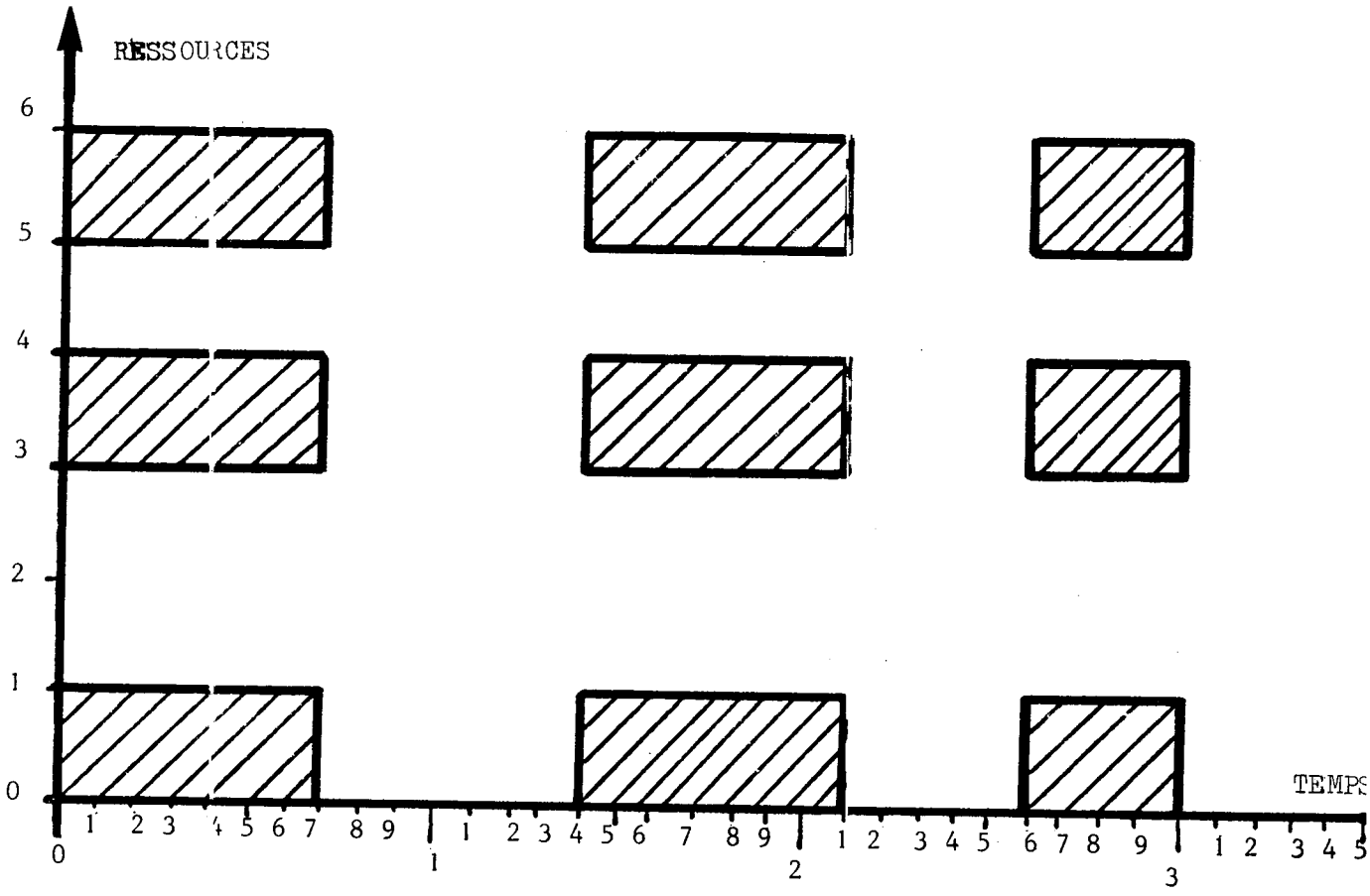


Figure 2 : Les trois activations de la tâche T1.

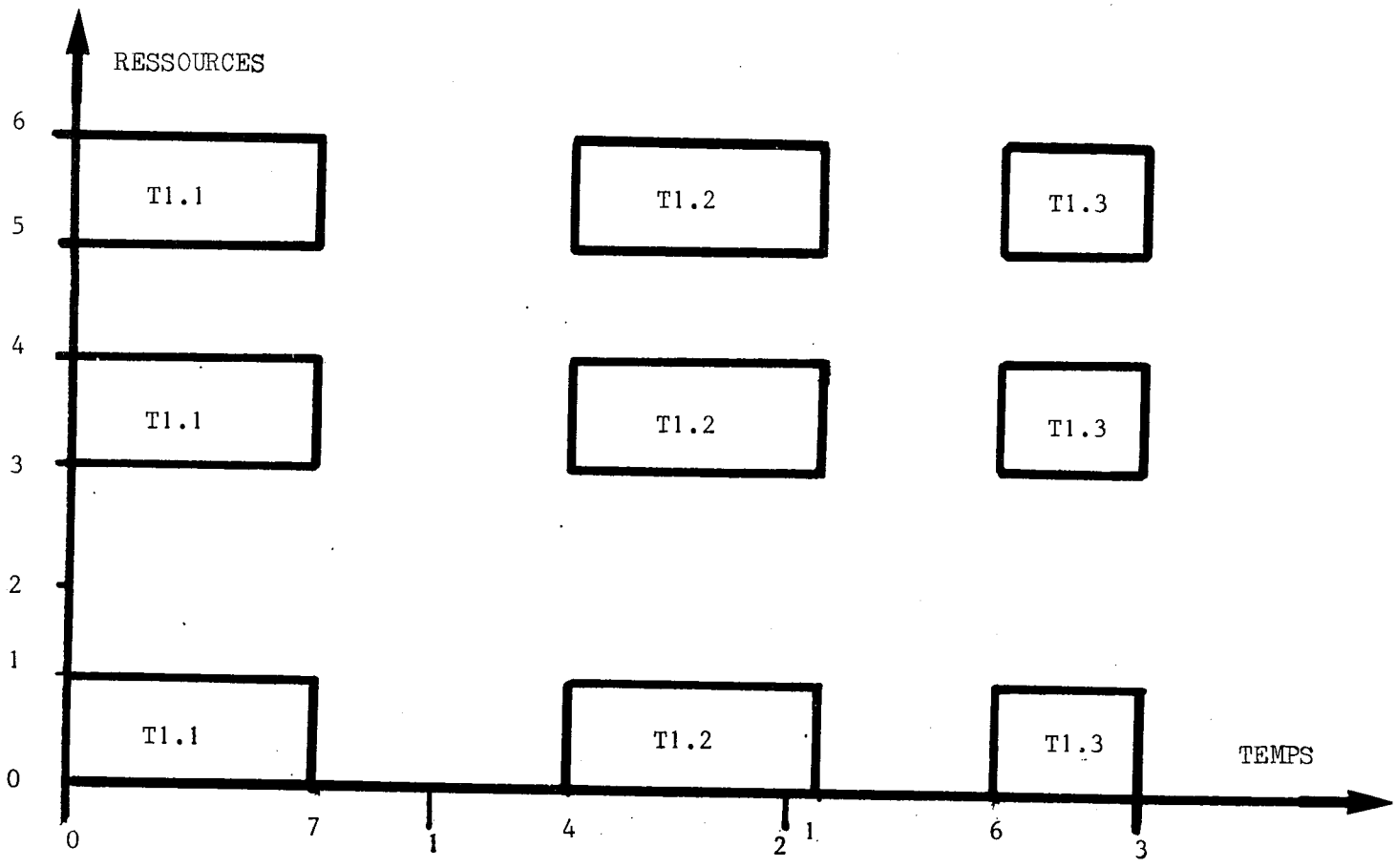
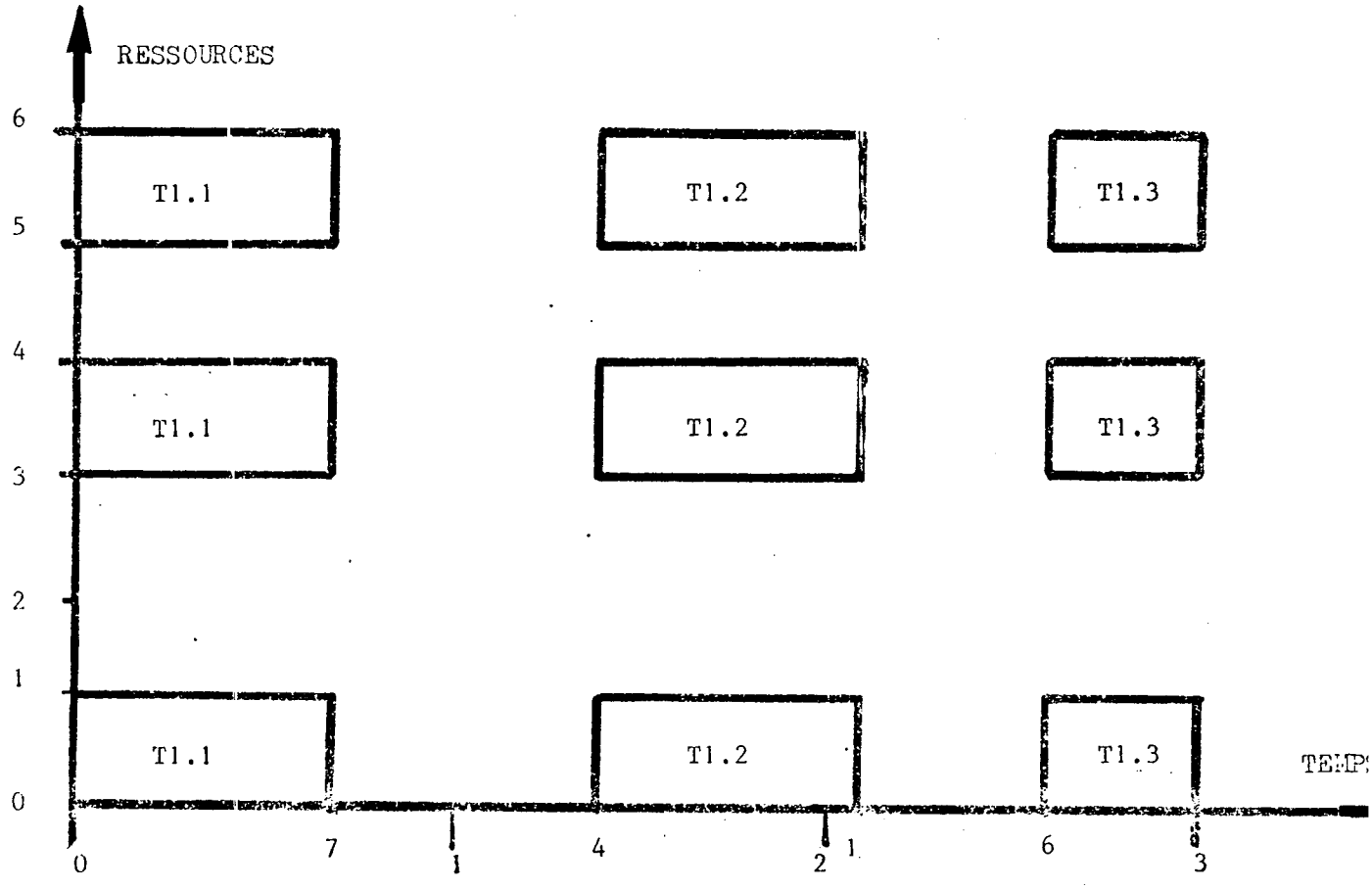


Figure 3 : Enumération des étapes du processus.



T1 **T1** T1

1ère activation
heure d'activation :
h(T1,1) = 0
durée de l'activation :
d(T1,1) = 7
ressources allouées :
M(T1,1) = (1,4,6)

2ème activation
heure d'activation :
h(T1,2) = 14
durée de l'activation :
d(T1,2) = 7
ressources allouées :
M(T1,2) = (1,4,6)

3ème activation
heure d'activation :
h(T1,3) = 26
durée de l'activation
d(T1,3) = 4
ressources allouées :
M(T1,3) = (1,4,6)

Figure 4 : Interprétation du diagramme.

Agissant de même pour toutes les autres tâches nous obtenons un ensemble de trois matrices :

- une matrice d'activation (Fig. 5) ;
- une matrice des durées (Fig. 6) ;
- une matrice d'allocation (Fig. 7).

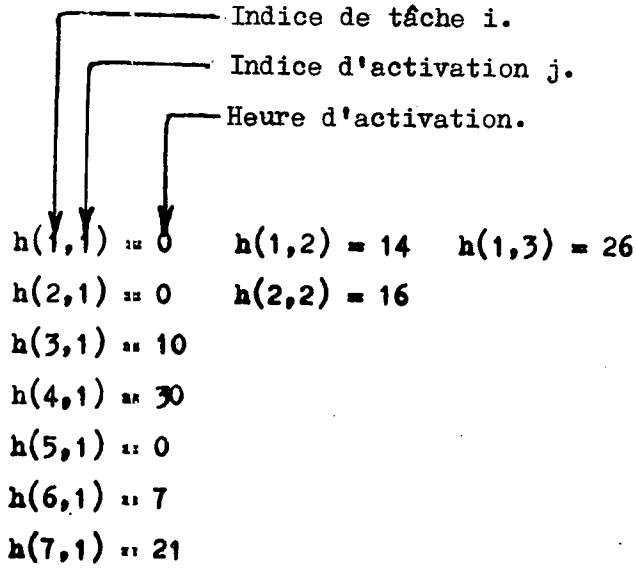


Figure 5 : Matrice d'activation.

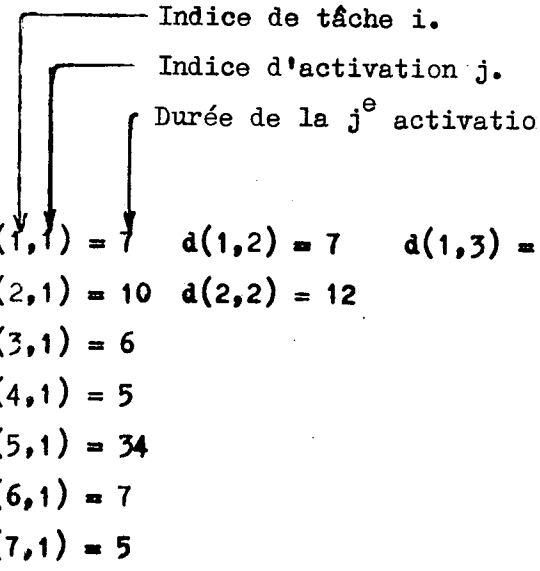


Figure 6 : Matrice des durées.

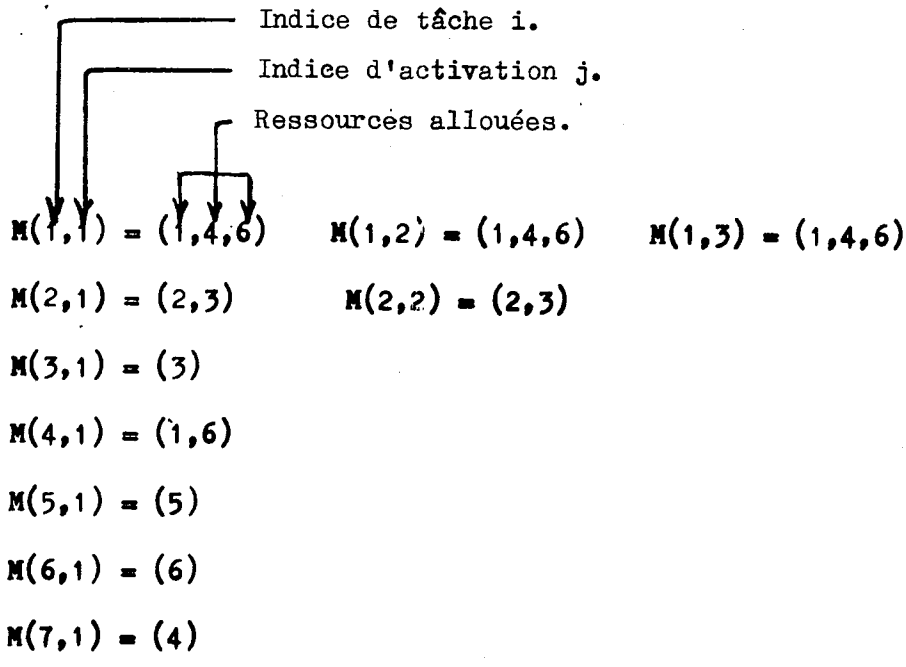


Figure 7 : Matrice d'allocation.

Nous avons relié tout diagramme à un ensemble de trois matrices. Mais la réciproque n'est pas vraie, tout ensemble de trois matrices ne peut être relié à un diagramme. De plus, un tel ensemble de matrices peut correspondre à un diagramme jugé inintéressant.

Notre but est donc d'examiner le minimum de matrices, et néanmoins trouver le meilleur ordonnancement, quel que soit le critère auquel meilleur se réfère.

Nous chercherons donc à définir un ensemble de contraintes, $\mathcal{C}$, tel que si l'on note $(\mathcal{M} \oplus \mathcal{M} \oplus \mathcal{M}) = \mathcal{E}$ l'ensemble des triplets, et $\mathcal{E}/\mathcal{C}$ l'ensemble de ceux qui vérifient les contraintes, alors :

1) L'ensemble des diagrammes solutions possibles soit en bijection avec $\mathcal{E}/\mathcal{C}$.

2) Le cardinal de $\mathcal{E}/\mathcal{C}$ soit aussi faible que possible, et si possible égal à 1.

Et nous chercherons, de même, un algorithme qui nous permette de trouver tous les éléments de $\mathcal{E}/\mathcal{C}$, si possible par un système de génération qui ne produise que les éléments de $\mathcal{E}/\mathcal{C}$.

Nous allons maintenant essayer de voir comment déterminer l'ensemble $\mathcal{C}$ de contraintes correspondant à un problème donné, en se basant sur des considérations simples sur les diagrammes recherchés.

III. 3. DETERMINATION DE QUELQUES CONTRAINTES.

Nous avons montré en II que l'ensemble des diagrammes était en bijection avec un sous-ensemble des triplets de matrices à deux dimensions à valeurs dans les entiers ou les listes d'entiers.

Si l'on considère des ressources non préemptives, une tâche ne peut être arrêtée, aussi, il n'est plus nécessaire de considérer deux dimensi L'ensemble est alors réduit à celui des triplets de matrices à une dimension valeurs dans :

$$\left|\begin{array}{l} \{DO\} \subset \{\{h(i)\}_*\{d(i)\}_*\{M(i)\}\} \\ \{h(i)\} = \mathbb{N}^t \\ \{d(i)\} = \mathbb{N}^t \\ \{M(i)\} = (2^r)^t = \left(\sum_{k=1}^{r} \binom{k}{r}\right)^t \end{array}\right.$$

avec : { } ensemble de,

$\mathbb{N}$ ensemble des entiers naturels,

t nombre de tâches,

r nombre de ressources,

DO Diagramme d'Occupation.

Dans le cadre du temps réel, la date d'activation est bornée par la date critique :

$$\left| h(i) \leq hdl(i) - d(i). \right.$$

Si l'on considère que les demandes de ressources sont faites sous la forme :

je veux W(i,k) machines de l'ensemble C(i,k,h(i)), ce pour les $||R/E_{i,h(i)}||$ ensembles de machines ($k = 1 \ldots N/E_i$); ce, pour toutes les tâches ($i = 1 \ldots t$).

C'est-à-dire qu'une tâche définit à l'instant de son activation (h(i)) une relation d'équivalence ($E_{i,h(i)}$) qui partitionne l'espace des ressources R en $||R/E_{i,h(i)}||$ classes d'équivalences C(i,k,h(i)) ;

alors, M(i) ne peut prendre ses valeurs que dans l'espace des combinaisons possibles au nombre de :

$$\prod_{k=1}^{||R/E_{i,h(i)}||} \begin{bmatrix} W(i,k) \\ \\ ||C(i,k,h(i))|| \end{bmatrix}$$

on lui donne W(i,k) ressources parmi les $||C(i,k,h(i)||$ de la classe C(i,k,h(i)), ce pour les $||R/E_{i,h(i)}||$ classes d'équivalences :

$$\left| M(i) \in \left\{ 1, 2, \ldots, \prod_{k=1}^{||R/E_{i,h(i)}||} \begin{bmatrix} W(i,k) \\ \\ ||C(i,k,h(i))|| \end{bmatrix} \right\} \right.$$

Si la durée des tâches est indépendante du choix des ressources ou des autres tâches alors :

$$d(i) = \text{cste}.$$

La figure 10 résume ces premiers résultats.

Un ensemble $\mathcal{C}_1$ de contraintes a été défini, tel que :

$$\mathcal{E}/\mathcal{C} \subseteq \mathcal{E}/\mathcal{C}_1 \subset \mathcal{E}.$$

Dans Ordon un deuxième ensemble de contraintes $\mathcal{C}_2$ est défini, tel q

$$\mathcal{C}_1 \cup \mathcal{C}_2 \subseteq \mathcal{C},$$

c'est-à-dire :

$$\mathcal{E}/\mathcal{C} \subseteq \mathcal{E}/\mathcal{C}_1 \cup \mathcal{C}_2,$$

de plus, il est prouvé que :

$$\mathcal{C}_1 \cup \mathcal{C}_2 = \mathcal{C}$$

c'est-à-dire :

$$\mathcal{E}/\mathcal{C} = \mathcal{E}/\mathcal{C}_1 \cup \mathcal{C}_2.$$

C'est le but des chapitres III, IV, V de prouver que dans le cas où classes de ressources sont constantes par rapport aux tâches et par rapport au temps, l'ensemble des diagrammes solutions est en bijection avec un sous ensemble des t! premiers entiers, qu'il est possible par un algorithme de générer exactement [1].

On y trouve les démonstrations, et la description de l'algorithme; tandis qu'en annexe, on trouvera la description de la version ALGOL-60 de l'algorithme [2].

$$\{\text{Gantt Charts}\} \subset \left\{ \{h(i,j)\},\ \{d(i,j)\},\ \{M(i,j)\} \right\}_{\substack{i=1,\ldots,t \\ j=1,\ldots}}$$

Si les ressources sont non préemptives :

$$\{\text{Gantt Charts}\} \subset \left\{ \{h(i)\},\ \{d(i)\},\ \{M(i)\} \right\}_{\substack{i=1,\ldots,t \\ j=1}}$$

$\{h(i)\} = N^t$

$\{d(i)\}$ = un ensemble fixe de valeurs

$$\{M(i)\} = (2^r)^t = \left(\sum_{k=1}^{r} \binom{k}{r} \right)^t$$

Si les durées sont fixes (pas de set-up time) :

$\{d(i)\}$ = un ensemble fixe de t constantes.

Dans le cadre d'un système Temps-Réel :

$$\{h(i)\} = \bigotimes_{i=1}^{m} \{0,\ldots,hdl(i) - d(i)\}.$$

Dans le cas où les demandes de ressources sont fixes :

$$\{M(i)\} = \bigotimes_{i=1}^{t} \left\{ 0,\ldots, \prod_{k=1}^{||R/E_{i,h(i)}||} \begin{pmatrix} W(i,k) \\ ||C(i,k,h(i))|| \end{pmatrix} \right\}$$

Figure 8 : Récapitulatif des propriétés et contraintes.

Pour ce travail, les résultats sont donnés dans le cas où :

1°) $a \in C(i,k,t)$ et $a \in C(i',k',t')$

entraîne

$C(i,k,t) = C(i',k',t')$

c.à.d. les classes sont indépendantes des tâches et temps.

$C(i,k,t) = C(k)$.

2°) $h(i)$ ne dépend que de :

- i,
- relations de précédence,
- l'occupation des ressources.

Nous ne prenons pas en compte :

- des intervalles fixes entre deux tâches,
- des modifications des caractéristiques des ressou

Dans ce cas nous démontrons :

$\{\text{Gantt Charts}\} \supset \{\text{Active Gantt Charts}\}$

$\{\text{Active Gantt Charts}\} \subset \{0,\ldots,t!\}$

Figure 9 : Récapitulatif des hypothèses faites pour Ordon.

III. 4. NECESSITE D'UNE RECHERCHE SYSTEMATIQUE

Les trois premiers paragraphes ont servi à présenter un ensemble de réflexions aptes à limiter l'espace des solutions possibles. Ces réflexions n'ont pas permis de définir une application simple qui du problème fasse passer à une solution. C'est pourquoi, nous allons envisager l'utilisation d'une procédure de recherche systématique, qui permette de parcourir l'espace des solutions possibles afin de déterminer celles qui conviennent.

III. 5. UNE METHODE DE PARCOURS DE L'ESPACE DES SOLUTIONS POSSIBLES

Il existe de nombreuses méthodes pour énumérer les éléments d'un ensemble fini. Le but recherché est toujours, outre l'énumération, l'obtenti de critères propres à éliminer le plus tôt possible les éléments qui ne sont pas solution, et propres à orienter la recherche de façon à arriver le plus tôt possible sur un élément solution. Selon la méthode et les critères, on obtient une procédure par élimination, un heuristique de recherche, un heuris tique de transformation, une recherche orientée, ou un heuristique de constru tion.

Une des méthodes structurées par élimination consiste à établir une bijection entre les éléments de l'ensemble des solutions possibles, et l'ense ble des feuilles d'un arbre, puis de définir un ensemble de propriétés qui si elles sont vraies pour un nœud de l'arbre le sont également pour toute feuil issue de ce nœud. A l'aide d'un quelconque algorithme on parcourt ensuite l'arbre, sachant que si une de ces propriétés est vérifiée pour un nœud, ell l est pour toute sa descendance, et qu'il est donc inutile de considérer chac des nœuds la composant.

III. 6 . APPLICATION AU PROBLEME D'ORDON

Si l'on tient compte pour Ordon de certains facteurs complémentaires on peut ramener le cardinal de l'espace des solutions possibles tel que défini aux paragraphes 1,2,3 à un nombre plus petit, quoique considérable, donné par la formule :

$$P_\tau = \sum_{i=1}^{|\tau|} P_{\tau,i} \qquad \text{si } |\tau| = 0,\ P_\tau = 1$$

$$P_{\tau,i} = \left\{ \prod_{j=1}^{|N/E_i|} \begin{pmatrix} \omega(i,j) \\ \\ |C(i,j)| \end{pmatrix} \right\} \times P_{\tau-\{i\}}$$

(Voir [1]).

avec $P_\tau = |\tau|!$ si $|C(i,j)| = \omega(i,j)$

Pour Ordon, nous cherchons à atteindre deux buts, d'une part, ramener le cardinal de l'espace des solutions possibles à n! ; d'autre part, fournir une méthode de recherche qui évite d'avoir à considérer ces n! solutions possibles.

Pour ce faire, nous allons considérer le premier point comme démontré définir une application qui à toute permutation d'ordre n associe un ordonnancement, puis associer à l'ensemble des permutations d'ordre n son arbre naturel étendre l'application aux nœuds de cet arbre, définir la réciproque de l'application étendue et déterminer certaines propriétés qui caractérisent les nœuds appartenant à l'image réciproque de l'image de l'arbre. Ces propriétés permettent d'éliminer certaines feuilles et certains noeuds, ce qui rend l'application bijective entre l'arbre réduit et l'image de l'arbre (≡ l'image de l'arbre réduit). Nous montrons ensuite que tout ordonnancement utile figure une fois et une seule dans l'image de l'arbre réduit ; prouvant à forciori le premier point laissé sans démonstration.

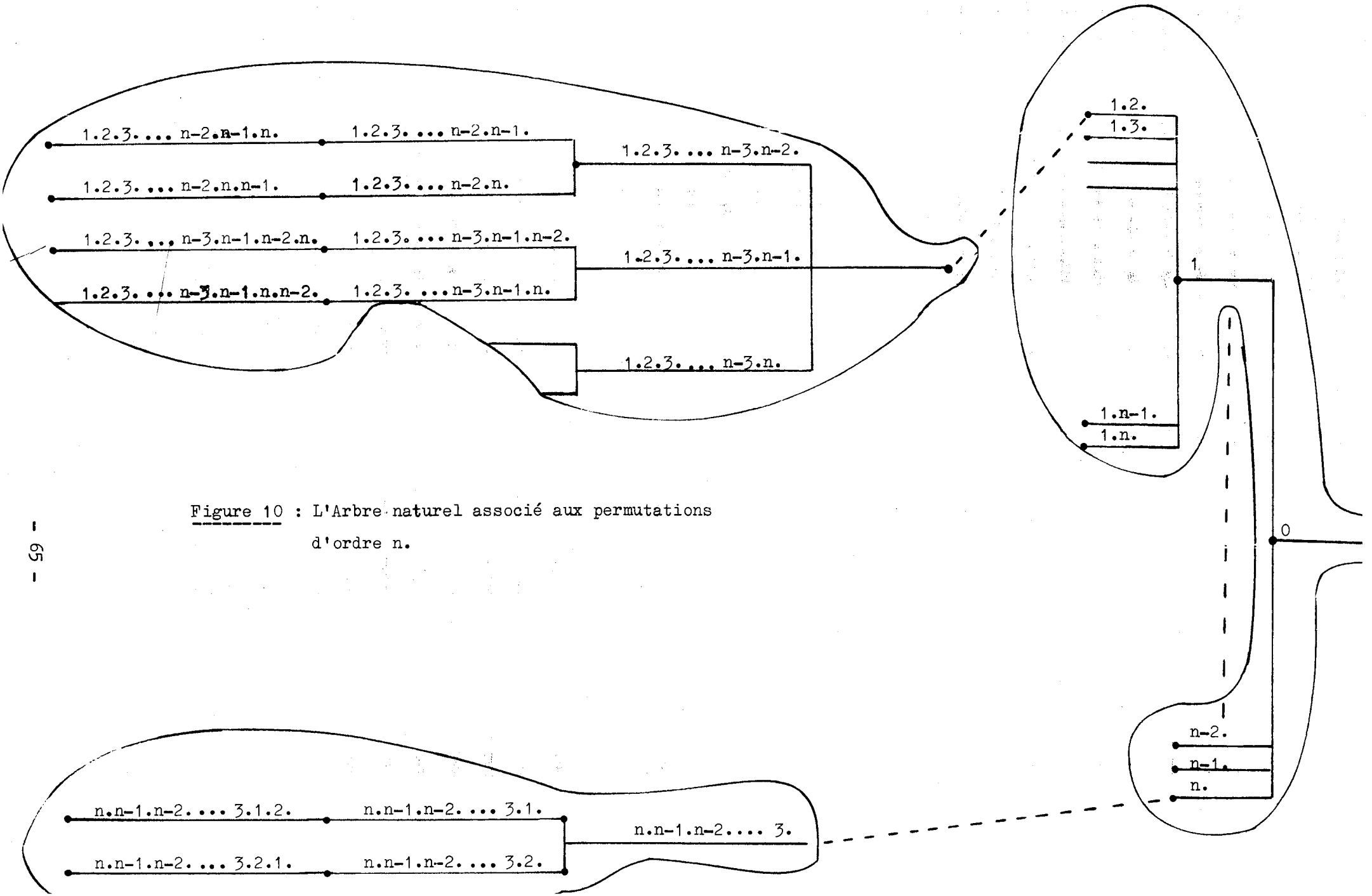


Figure 10 : L'Arbre naturel associé aux permutations d'ordre n.

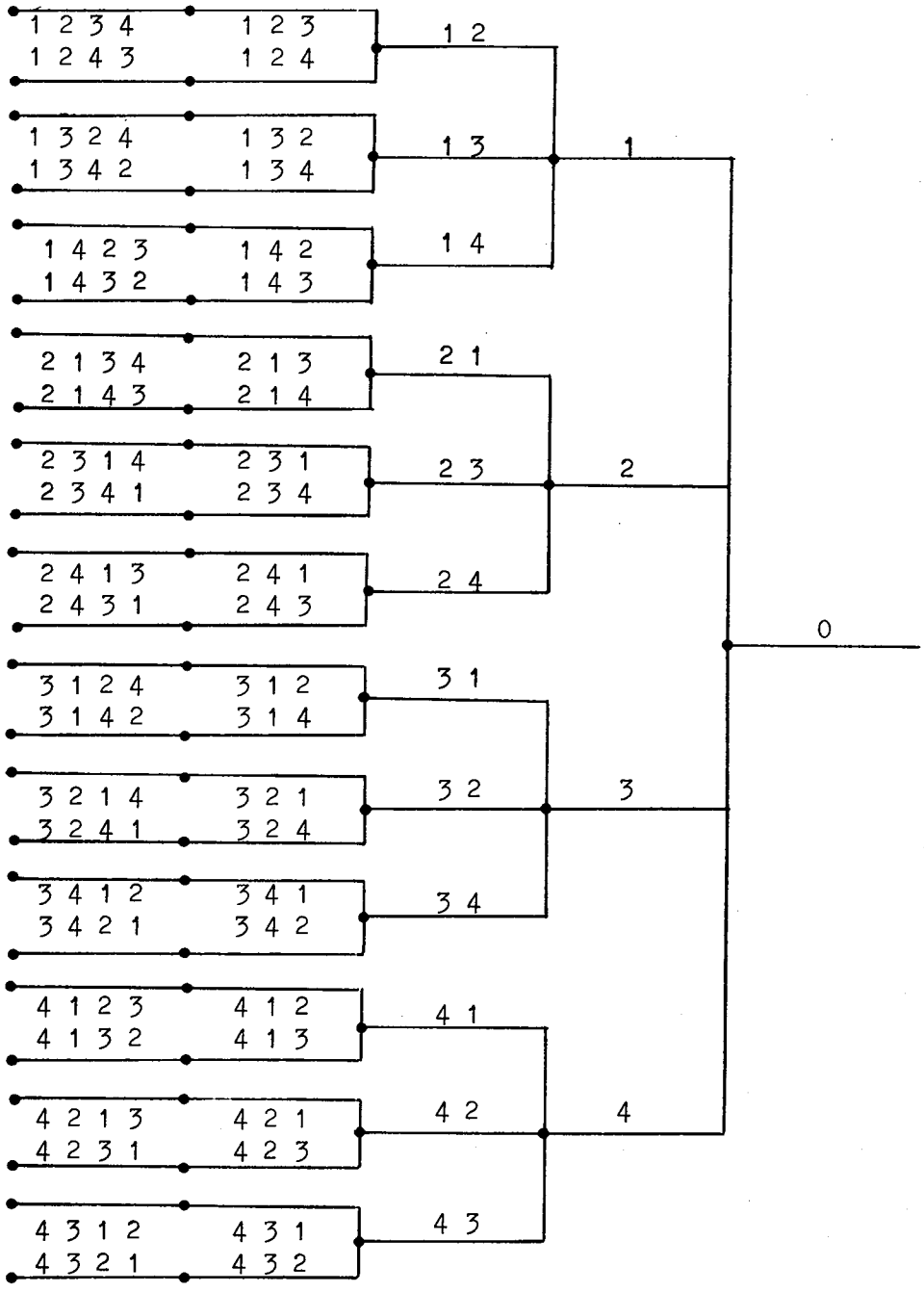


Figure 11 : L'Arbre naturel associé aux permutations d'ordre 4.

P espace des permutations.

O espace des ordonnancements.

$\overline{P}$ espace des permutations augmenté de l'arbre naturel.

$\overline{O}$ espace des ordonnancements et ordonnancements partiels.

SHP fonction de P → O étendue à $\overline{P}$ et $\overline{O}$.

C fonction de O → P étendue à $\overline{O}$ et $\overline{P}$.

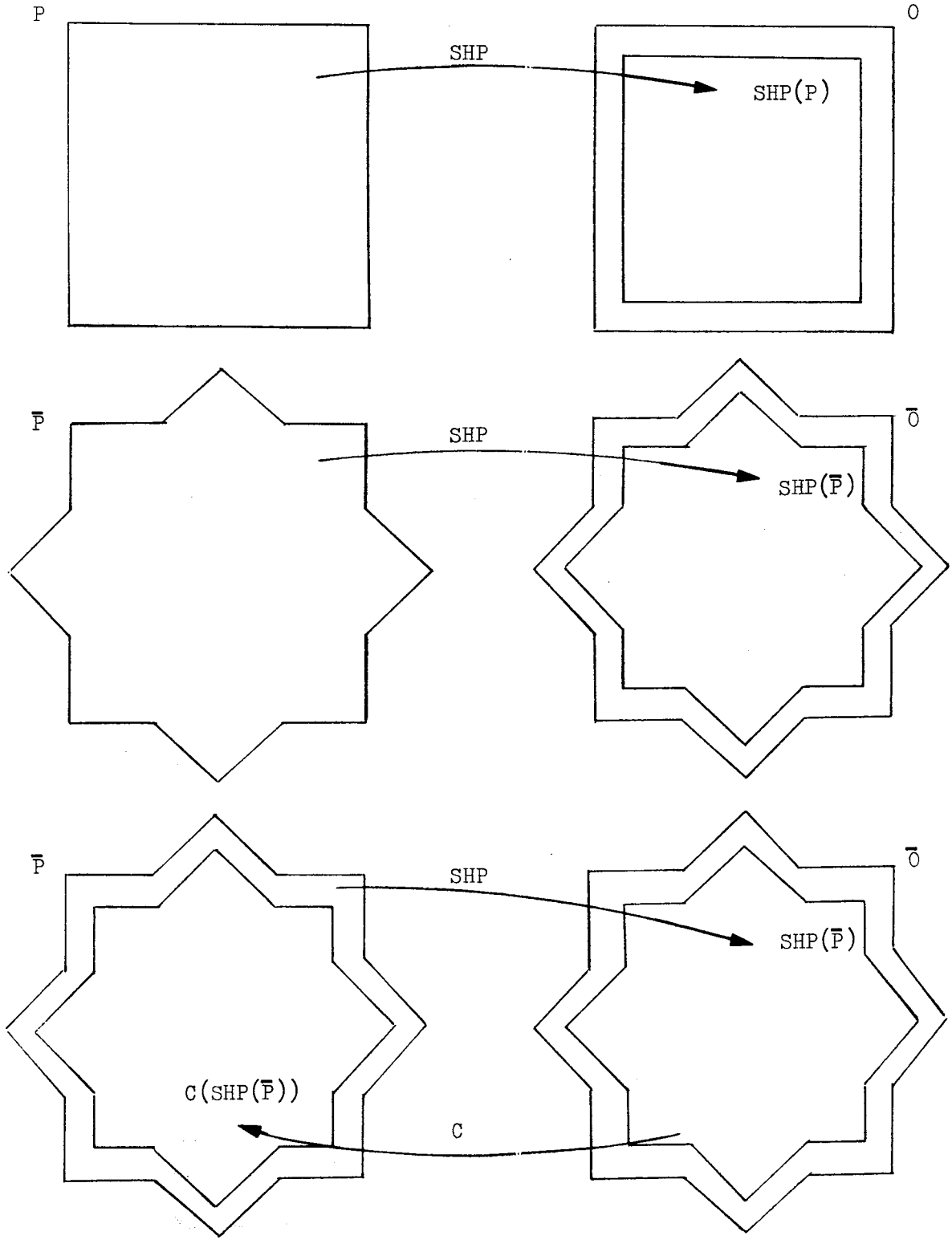


Figure 12 : Illustration de la méthode.

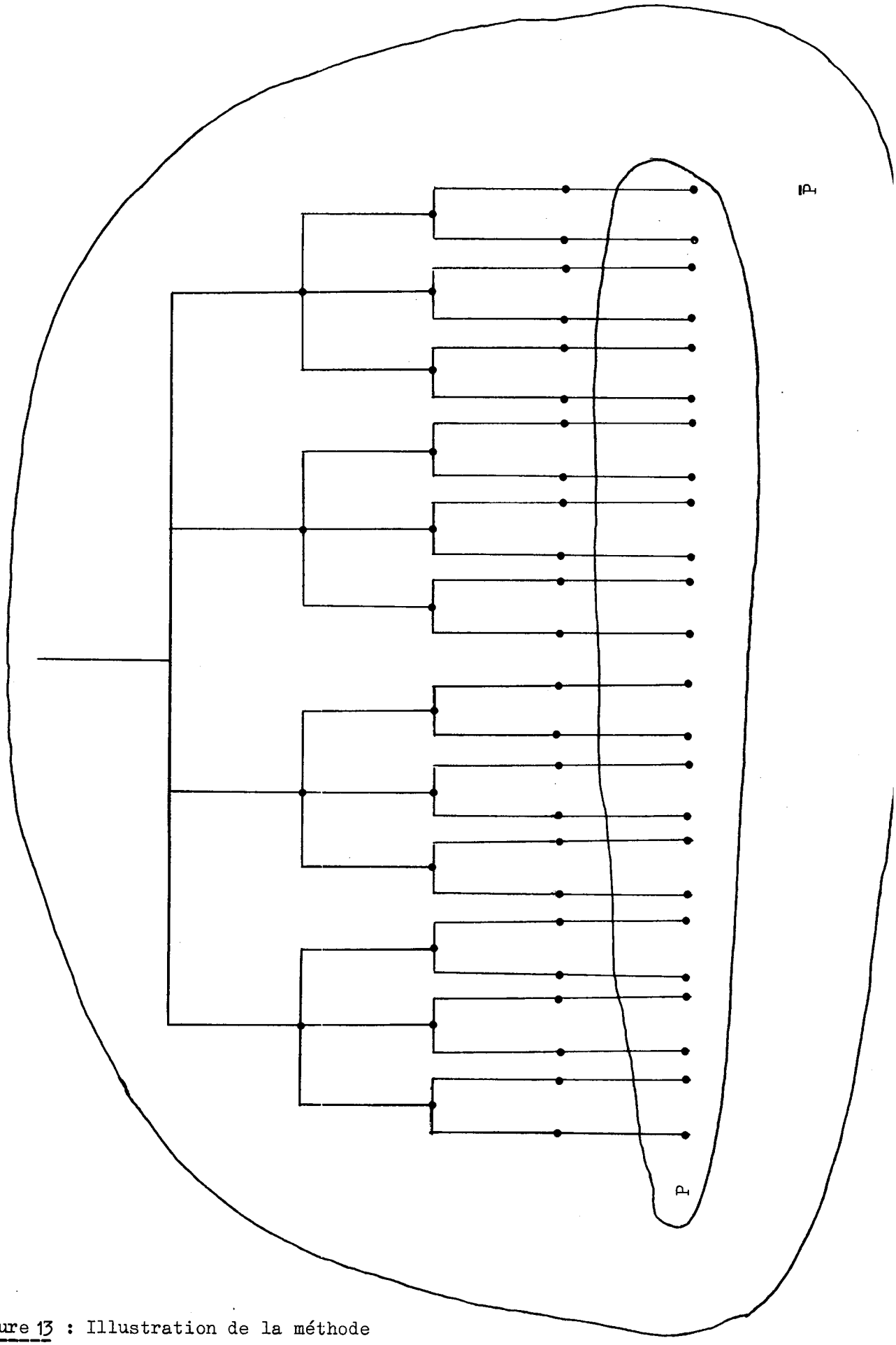


Figure 13 : Illustration de la méthode
Pour les permutations d'ordre 4 (1).

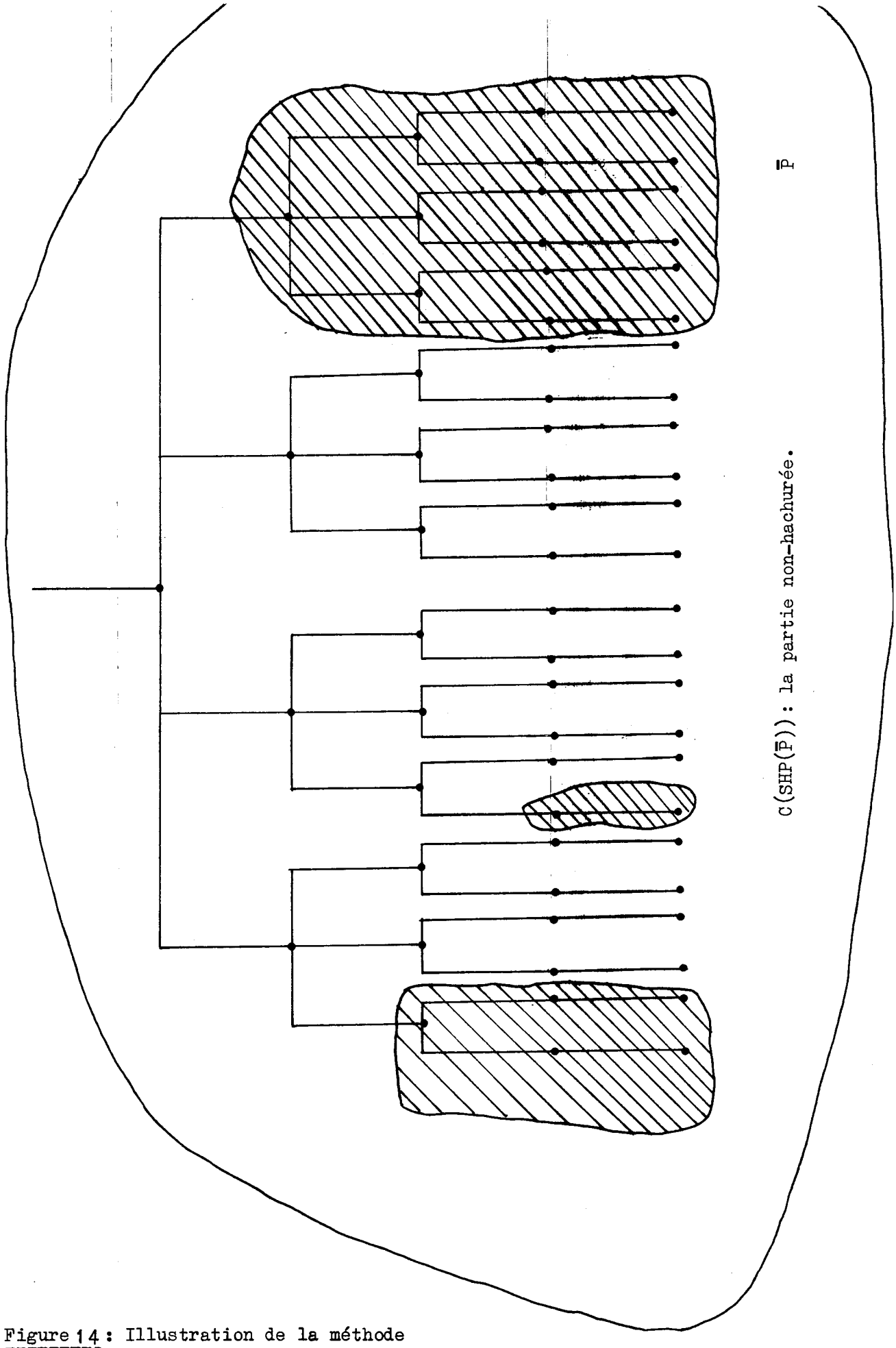


Figure 14 : Illustration de la méthode
Pour les permutations d'ordre 4 (2).

Nous montrons :

α) Que si un nœud n'appartient pas à $C(SHP(\bar{P}))$, alors aucun de ses descendants n'y appartient.

β) Qu'il existe des propriétés simples permettant de déterminer si un nœud appartient à $C(SHP(\bar{P}))$.

γ) Que la SHP est injective sur $C(SHP(\bar{P}))$.

δ) Que $C = SHP^{-1}$ resp. sur $SHP(\bar{P})$ et $C(SHP(\bar{P}))$

ε) Que tout ordonnancement utile appartient à $SHP(C(SHP(\bar{P})))$.

φ) Que tout ordonnancement appartenant à $SHP(C(SHP(\bar{P})))$ est utile.

D'où la méthode qui consiste à étudier les nœuds de l'arbre naturel en partant de la racine, si le nœud appartient à $C(SHP(\bar{P}))$, (ce que l'on détermine grâce à β) on continue, si non, on élimine ce nœud et tous ses descendants (grâce à α). On est ainsi sûr d'avoir envisagé tous les plannings utiles (ε), que les plannings utiles (φ) et une fois et une seule (γ).

Tableau 1 : Résumé de la démarche.

REFERENCES.

[1] **JORRY A.** **"ORDON : a scheduling algorithm for real-time tasks on non-preemptive resources with deadline".**
SPECTRE Rapport Interne **IA/112 (march 74),**
and
I.R.I.A./LABORIA Rapport n°64.

[2] JORRY A. "ORDON : un algorithme pour l'ordonnancement de tâches Temps-Réel sur des ressources non-préemptives.
Notice de programme. Version ALGOL-60".
SPECTRE Rapport Interne IP/224. IRIA-LABORIA. (Décembre 197

ANNEXE

Introduction des notions d'état, de transition, de trajectoire.

Reprenons la figure 1, qui représente le diagramme d'occupation dans l'espace des phases des tâches (Etat ⊗ temps), et transposons la dans l'espace de phases des ressources, ce qui permet de voir se dessiner la matrice d'allocation. (Fig. 15).

Introduisons maintenant une tâche virtuelle, dite inaction, et reprenons la trajectoire des ressources dans ce nouvel espace (Fig. 16 et 17.) ceci ne peut être fait que parcequ'une ressource n'a qu'un élément pour définir son état, alors qu'une tâche a un vecteur d'état.

Cette notion de trajectoire introduit la partie duale d'Ordon, consacrée à l'expression matricielle des contraintes, et la définition d'algorithmes propres à extrémiser certaines fonctions dépendant de ces trajectoires (Voir les travaux de M. Depeyrot)[1].

---

(1) - "Approche barycentrique par transformation de Fourier d'une classe de problèmes de placement et de transport".
C.R. Acad. Sc. Paris t 279 (2 Décembre 1974). Série A. pp.823-826.

- "Une généralisation de la notion d'automate et applications".
Thèse soutenue à l'USMG (Grenoble) le 24 Juin 1975.

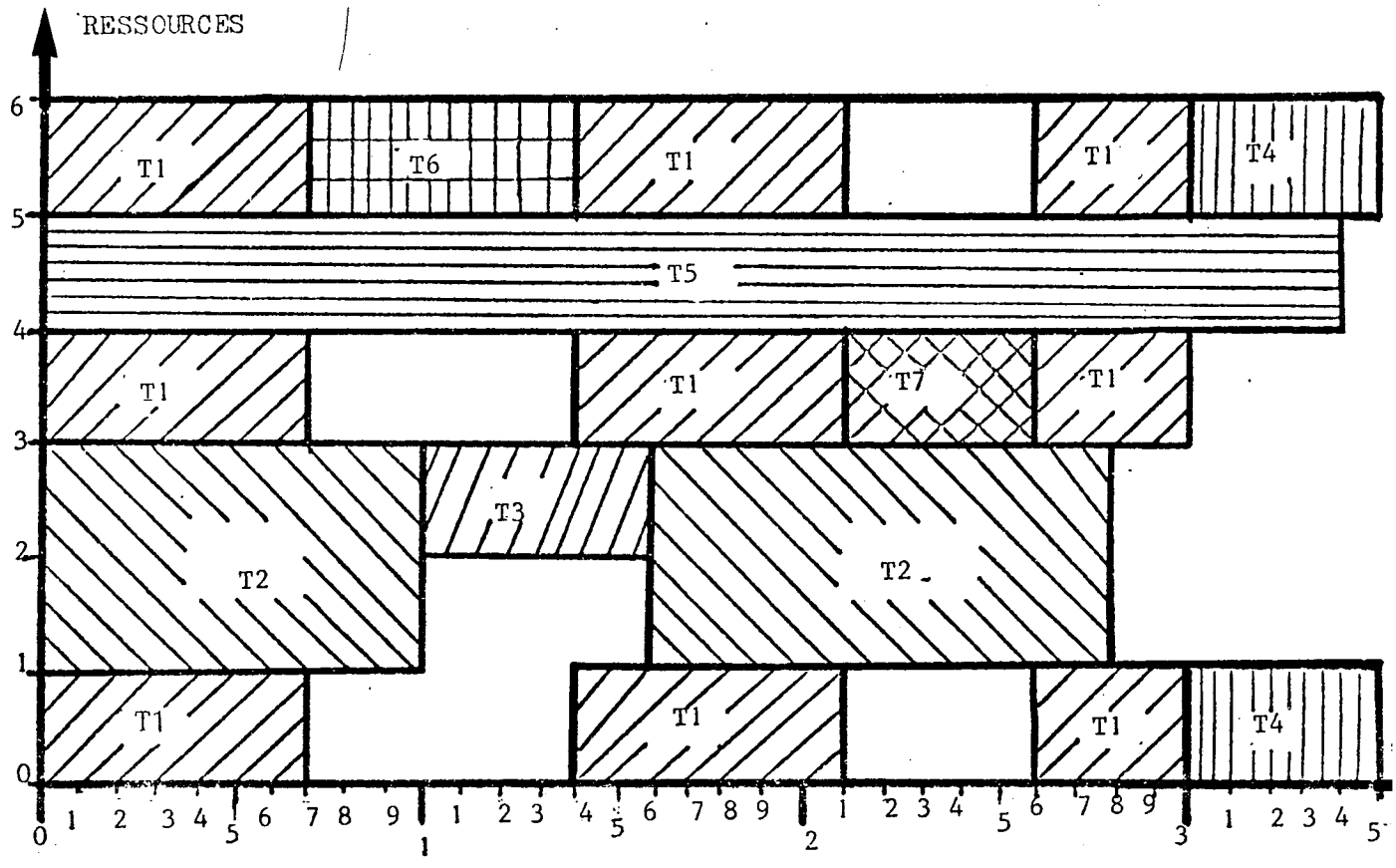


Figure 1 : Diagramme d'occupation de l'exemple (Rappel).

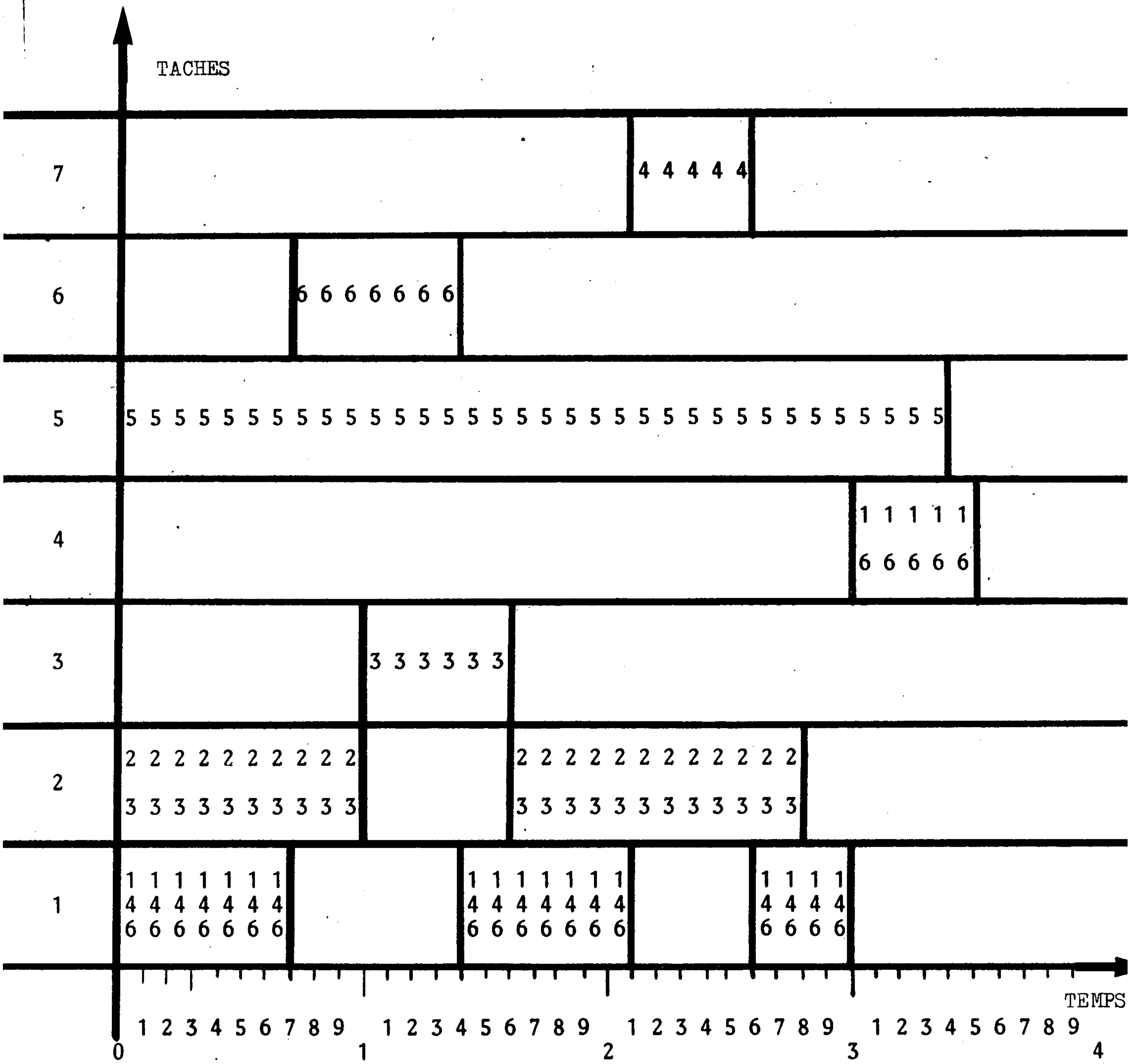


Figure 15 : Représentation des activations dans l'espace de phase des ressources.

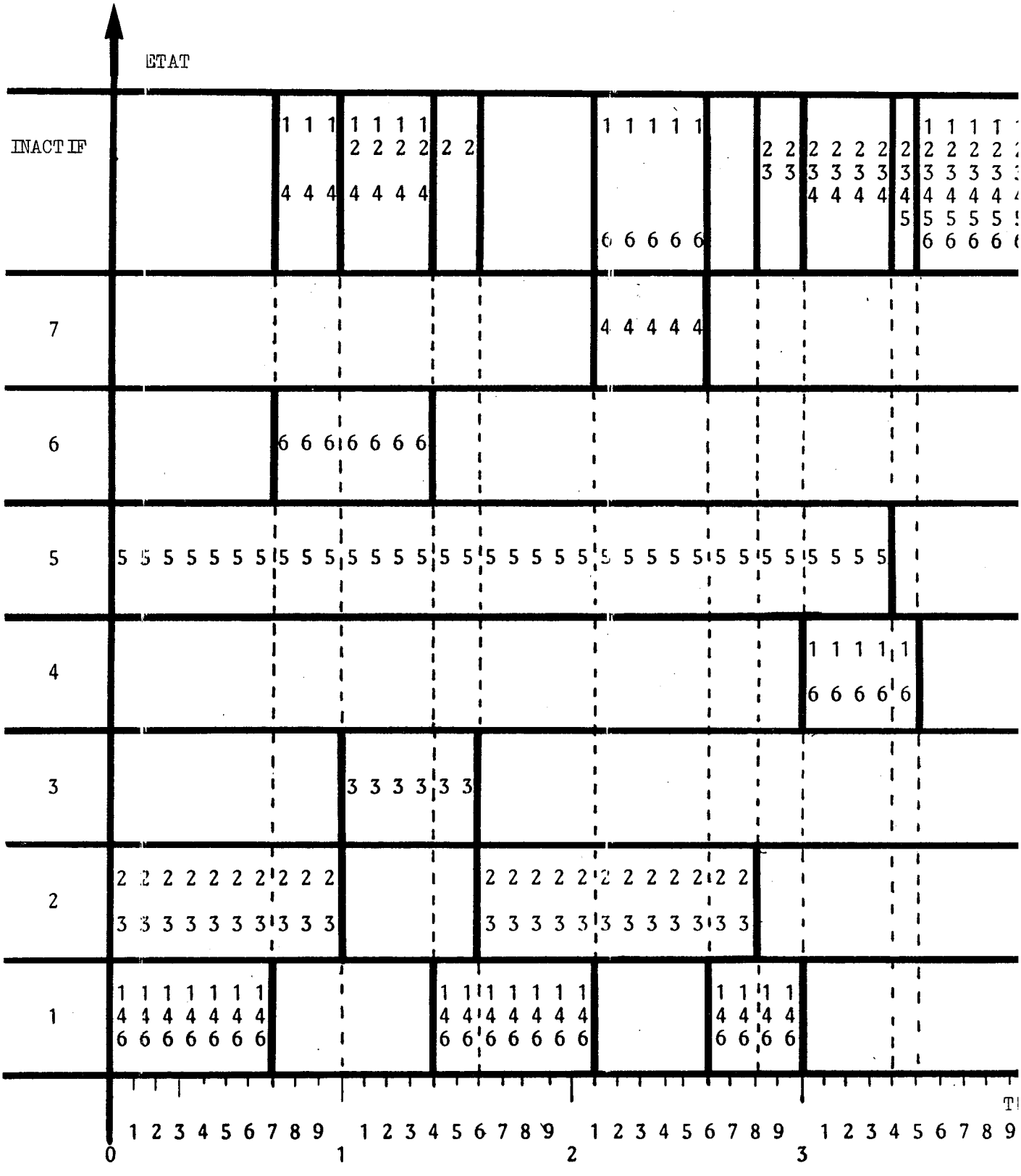


Figure 16 : Représentation de la trajectoire des six ressources.

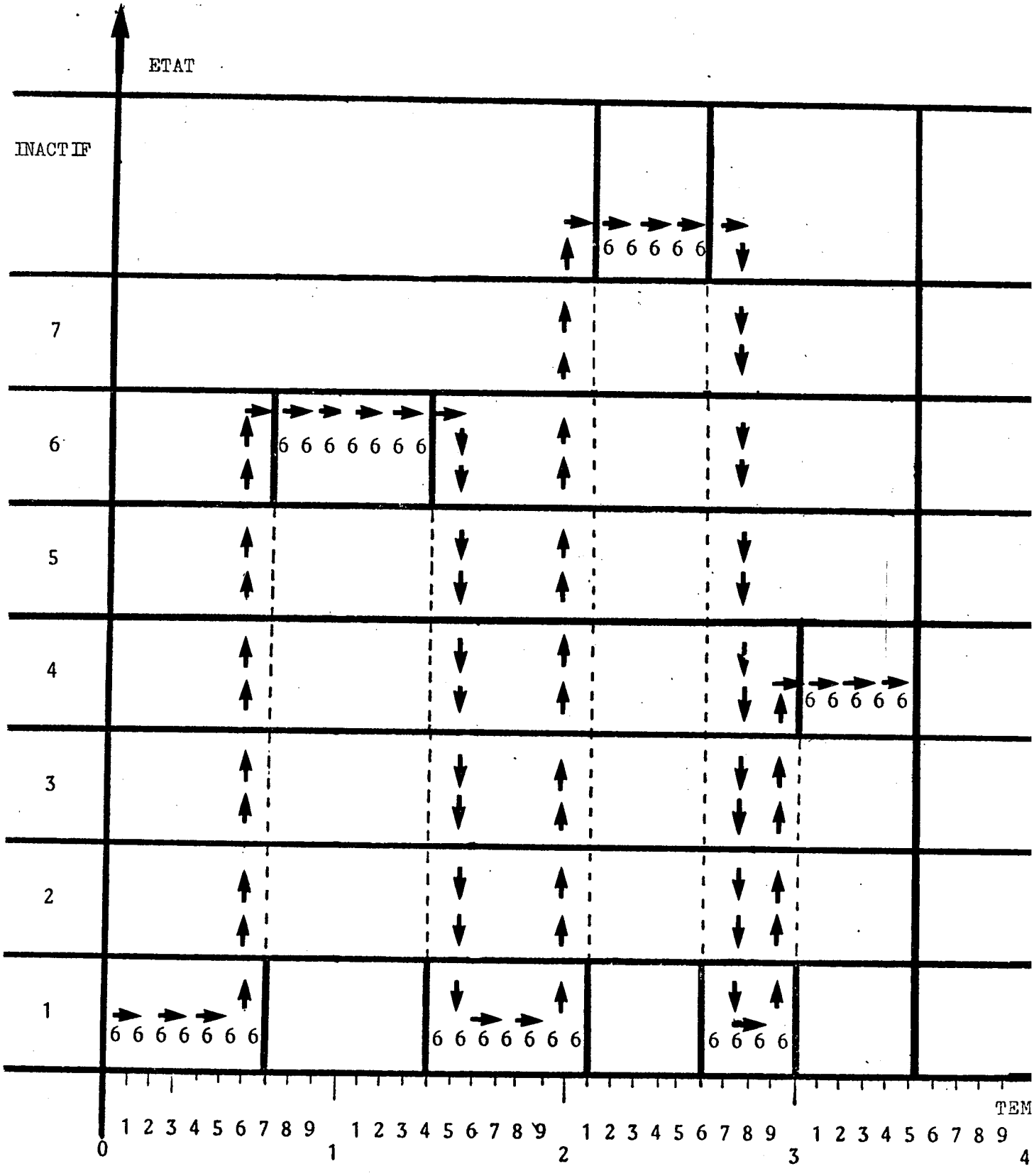


Figure 17 : Représentation de la trajectoire de la ressource six. (Détail de la figure 16).

Chapitre quatrième.

Un algorithme pour la résolution de ce problème :

ORDON.

TABLE DES MATIERES

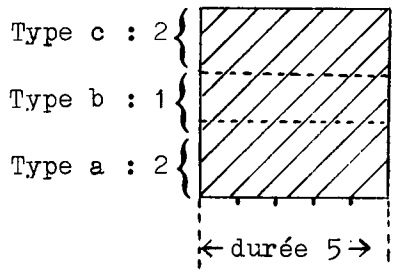
IV. 1. DEFINITION DE LA SHP

IV. 1. 1. Représentation des demandes.

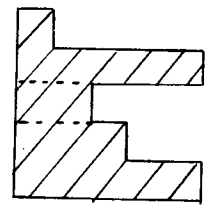
Soit un ensemble de n tâches, et un ensemble de ressources réparties en r types. Numérotons les n tâches de 1 à n. Considérons les demandes de ressources comme effectuées de la façon suivante :

- Pour chaque tâche i, pour chaque type de ressources j, la tâche indique le nombre $\omega(i,j)$ de ressources de ce type qu'elle désire.
- Dans un premier temps, les ressources seront allouées durant toute l'exécution. Nous étudierons par la suite le cas où les ressources sont toutes allouées au début de l'exécution et relâchées au fur et à mesure qu'elle se déroule ; puis le cas où demande et libération interviennent à tout instant de l'activité (V.5).

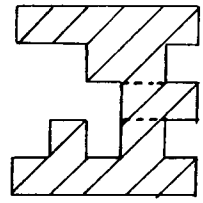
On peut représenter ces demandes de la façon suivante :



α) Allocation pour la durée de l'activité.



β) Relâche en cours d'activité.



γ) Allocation-libération sans contraintes.

Figure 1 : Représentation des demandes.

IV. 1. 2. Méthode d'ordonnancement.

Soit une permutation d'ordre n.

α) Considérons la tâche dont le numéro est le premier élément de la permutation.

β) Accordons lui au hasard un ensemble quelconque de ressources satisfaisant à sa demande. Ces ressources sont considérées par l'algorithme comm occupées de l'instant 0 jusqu'à la fin de la tâche (*).

γ) Considérons la tâche suivante (dont le numéro est l'élément suiva de la permutation). S'il reste assez de ressources pour satisfaire sa demande, procédons comme en β. Sinon calculons l'heure à laquelle un ensemble de ressources satisfaisant sera disponible et allouons lui les ressources libérées le plu tard possible avant cette date (*).

On continue ainsi jusqu'à la fin de la permutation.

IV. 1. 3. Illustrations.

Ces deux illustrations peuvent sembler fastidieuses, néanmoins elles mettent en évidence les deux caractéristiques de la méthode qui en constituent l'essentiel.

Illustration 1. La SHP (1ère partie).

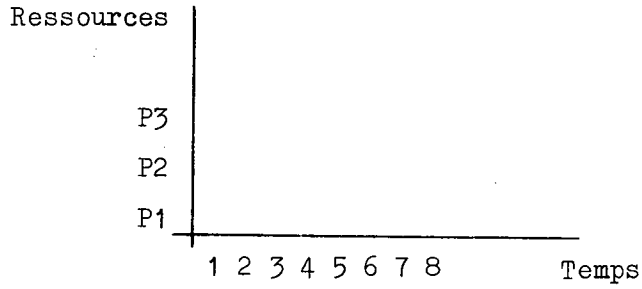
Considérons quatre tâches, un type de ressources, trois ressources de ce type et les demandes suivantes :

Tâche	ω(i,1)	Durée (i)
T1	1	1
T2	2	2
T3	2	3
T4	1	2

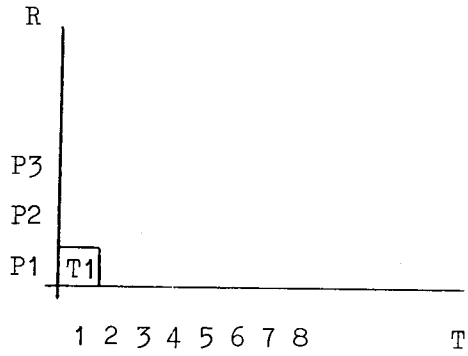
(*) Ces deux caractéristiques contituent l'essentiel de la méthode, et sont la base des propriétés qui seront démontrées au chapitre V. Elles sont illustré par les deux exemples qui suivent.

Prenons la permutation 1.2.3.4.

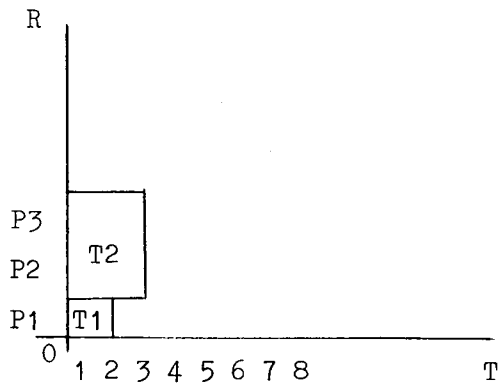
Construisons le planning :



α) Tous les processeurs sont libres on place T1 au hasard. Disons sur le processeur P1 :

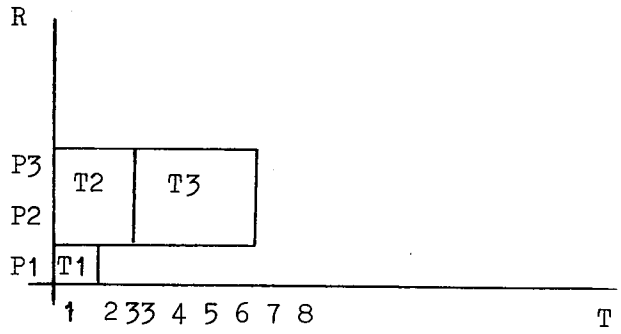


β) Il reste deux processeurs libres au temps 0. On place donc T2 sur ces deux processeurs :



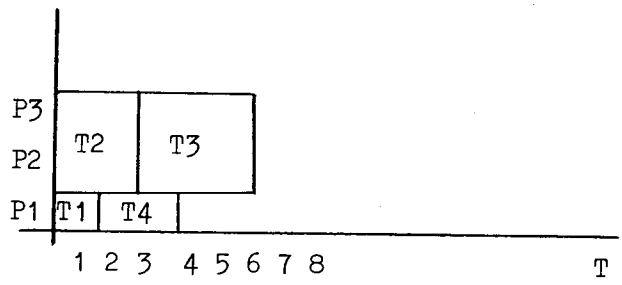
γ) Il n'y a deux processeurs libres qu'à l'instant t=2. T3 doit donc attendre jusqu'à cet instant.

A l'instant t=2 les trois processeurs sont libres ; on doit donc choisir. On prend les processeurs P2 et P3 puisque ce sont ceux qui sont libérés le plus tard avant cette date.



δ) Il n'y a un processeur libre qu'à l'instant t=1. T4 doit donc attendre jusqu'à cet instant.

A l'instant t=1 il n'y a qu'un processeur libre. C'est donc lui qui est alloué.



Remarque.

La façon dont T4 "est glissé sous T2 et T3" explique pourquoi il vaut mieux choisir au pas γ les processeurs P2 et P3 plutôt que P1 et P2 ou P1 et P3. Ce qui dans cet exemple semble être une évidence n'en est en fait pas une, et fait l'objet de cette étude et des démonstrations du chapitre V.

Illustration 2. La SHP (2^{de} partie).

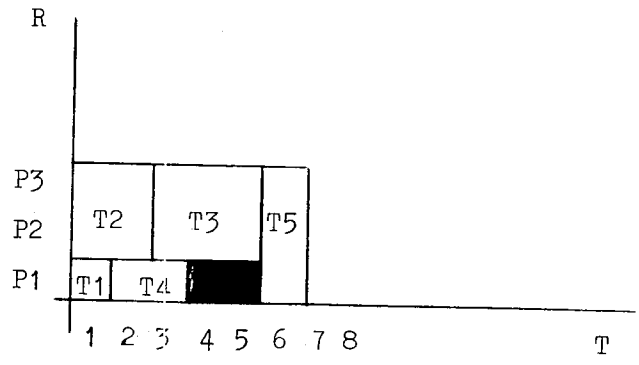
L'illustration 1 montre imparfaitement comment sont calculées les dates de libération des processeurs. Pour mieux l'illustrer nous allons ajouter deux autres tâches :

Tâche	ω(i,1)	Durée (i)
T5	3	1
T6	1	2

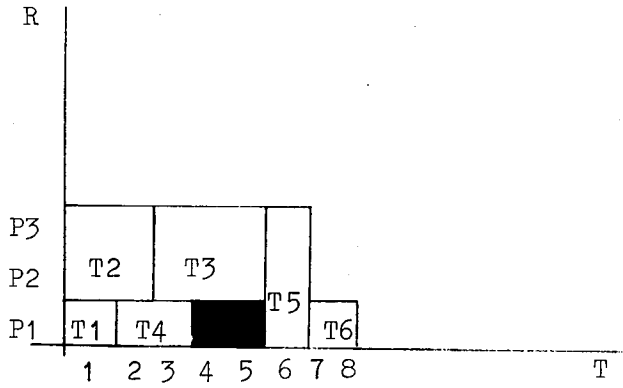
et considérer la permutation 1.2.3.4.5.6.

Les quatre premiers pas sont identiques à ceux de l'illustration 1.

ε) Il n'y a trois processeurs libres qu'à l'instant t=5. T5 doit donc attendre. Comme il n'y a que trois processeurs, il n'y a pas de choix possible.



φ) Cette méthode crée un "trou" dans le planning, mais ce trou ne sera pas récupéré, il est considéré comme "trou" ou temps mort et P1 est considéré comme libéré à l'instant t=6. T6 doit donc attendre bien que sa demande et sa durée lui permettent de se "glisser dans le trou". Nous verrons comment éviter de tels cas et prouverons que nous les évitons tous au chapitre V.



Nous verrons par la suite combien cette remarque est importante, puisqu'elle nous conduit à ne considérer que 3 (nombre de ressources) dates de libération, les dernières, et non 18 dates de libération et 18 durées de trous (nombre de ressources *(nombre de tâches -1 +1)).

pas la dernière

avant la première

IV. 2. EXTENSIONS DU PROBLEME

IV. 2. 1. Relations de précédence.

Si l'on considère que les tâches ne sont pas indépendantes, mais qu'il existe un graphe de précédence, l'heure de départ au plus tôt calculée n'est plus l'heure à laquelle il existe assez de ressources pour satisfaire la demande, mais le "sup" de cette heure et de l'heure de fin du dernier prédécesseur.

Dans ce cas, il est inutile de considérer toute permutation qui place une tâche avant l'un de ses prédécesseurs.

IV. 2. 2. Arrivées échelonnées.

Si l'on considère que les tâches ne sont pas toutes à disposition à l'instant 0 ; mais qu'elles arrivent à des dates diverses, l'heure de départ au plus tôt est alors le "sup" de l'heure à laquelle il existe assez de ressources disponibles, l'heure à laquelle se termine le dernier prédécesseur, et l'heure à laquelle la tâche arrive.

IV. 2. 3. Nota.

Les démonstrations seront faites dans le cas du problème étendu. Pour les problèmes plus simples, il suffit de considérer un graphe de précédence nul, ou des heures d'arrivée identiquement nulles, ou les deux à la fois.

IV. 3. UN ALGORITHME POUR LA GENERATION DES PERMUTATIONS RESPECTANT L'ORDRE DE PRECEDENCE [HELLER-LOGEMANN]

IV. 3. 1. Présentation.

Considérant l'arbre naturel associé au groupe des permutations d'ordre n Sn, on constate que si en un nœud l'ordre de précédence n'est pas respecté, il ne le sera pour aucun de ses descendants.

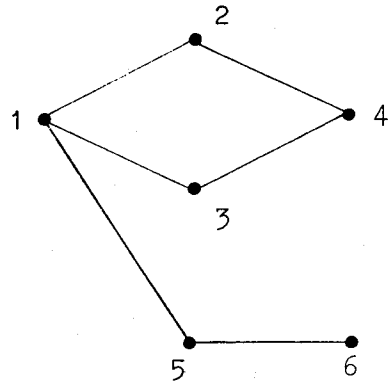
Considérons un nœud de l'arbre naturel où les relations de précédence sont respectées par la sous-permutation associée. Seuls de ses descendants immédiats sont intéressants ceux qui de même ne violent pas les liens de précédence, c'est-à-dire pour lesquels l'élément complémentaire est le numéro d'une tâche dont tous les prédécesseurs figurent dans la sous-permutation associée au nœud-père.

On peut baser un algorithme de génération de l'arbre résiduel dont tous les nœuds respectent la précédence sur cette remarque en agissant ainsi : de tous les successeurs d'un nœud, ne sont considérés que ceux dont l'élément complémentaire est le numéro d'une tâche dont tous les prédécesseurs figurent dans la permutation associée à ce noeud.

IV. 3. 2. Illustration par le traitement exhaustif d'un exemple.

Illustration 3. L'algorithme de HELLER-LOGEMANN.

Soit le graphe de précédence.

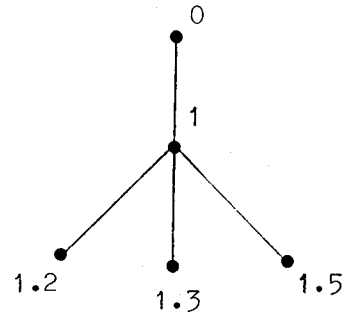


L'arbre résiduel associé se bâtit comme suit :

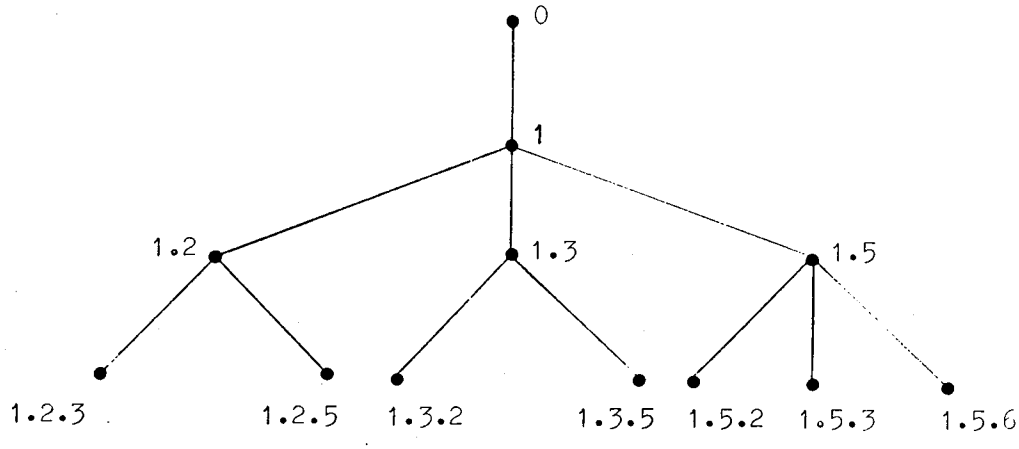
1) On positionne l'origine :

2) Seul 1 a tous ses prédécesseurs inclus dans la permutation 0, donc seul le noeud associé à la sous-permutation 1 est placé :

3) Maintenant 2,3 et 5 vérifient la propriété :



4) Seuls 3 et 5 ont tous leurs prédécesseurs inclus dans 1.2
2 et 5 1.3
2,3 et 6 1.5



5) Seuls 4 et 5 ont tous leurs prédécesseurs inclus dans 1.2.3

3 et 6 1.2.5

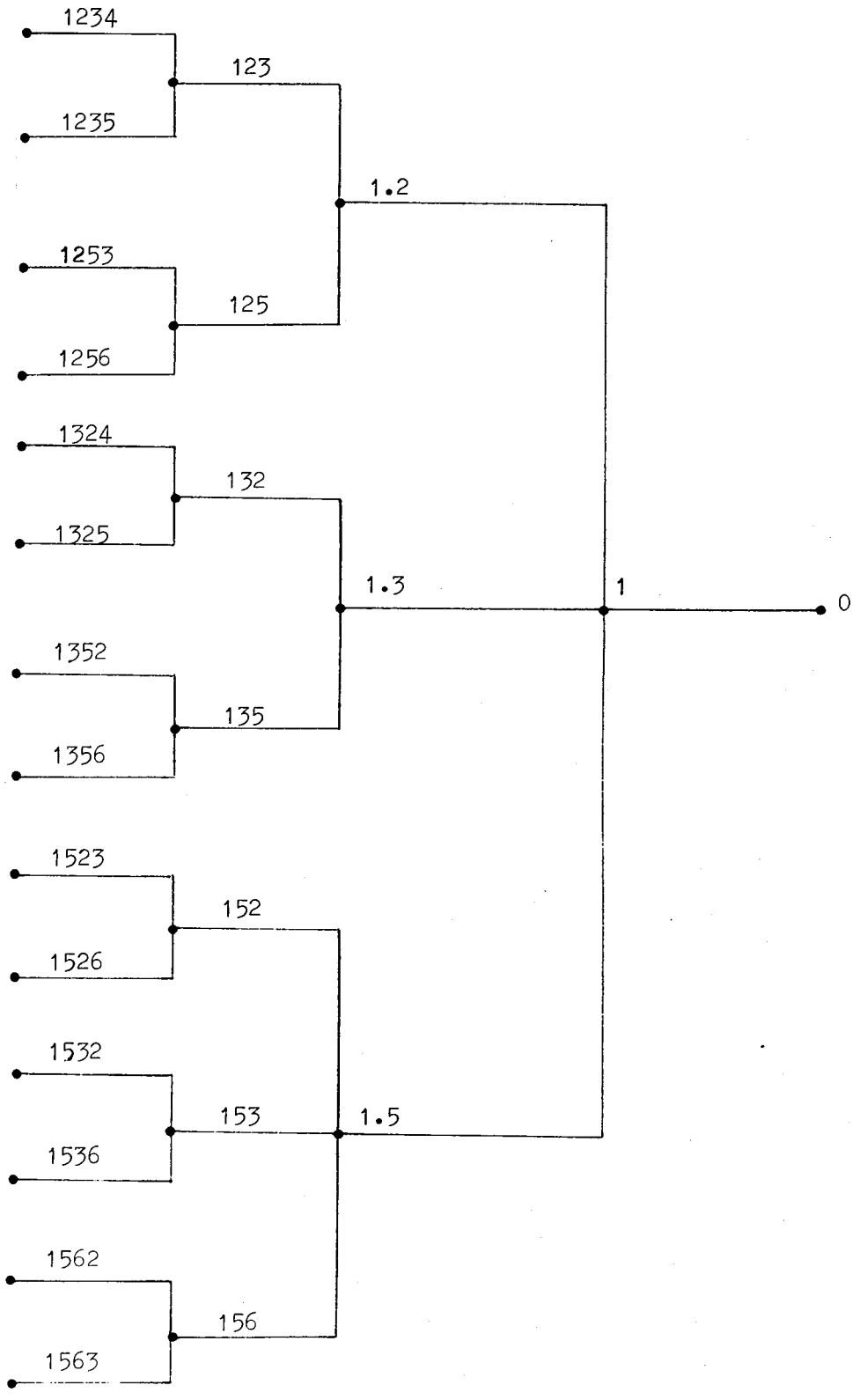
4 et 5 1.3.2

2 et 6 1.3.5

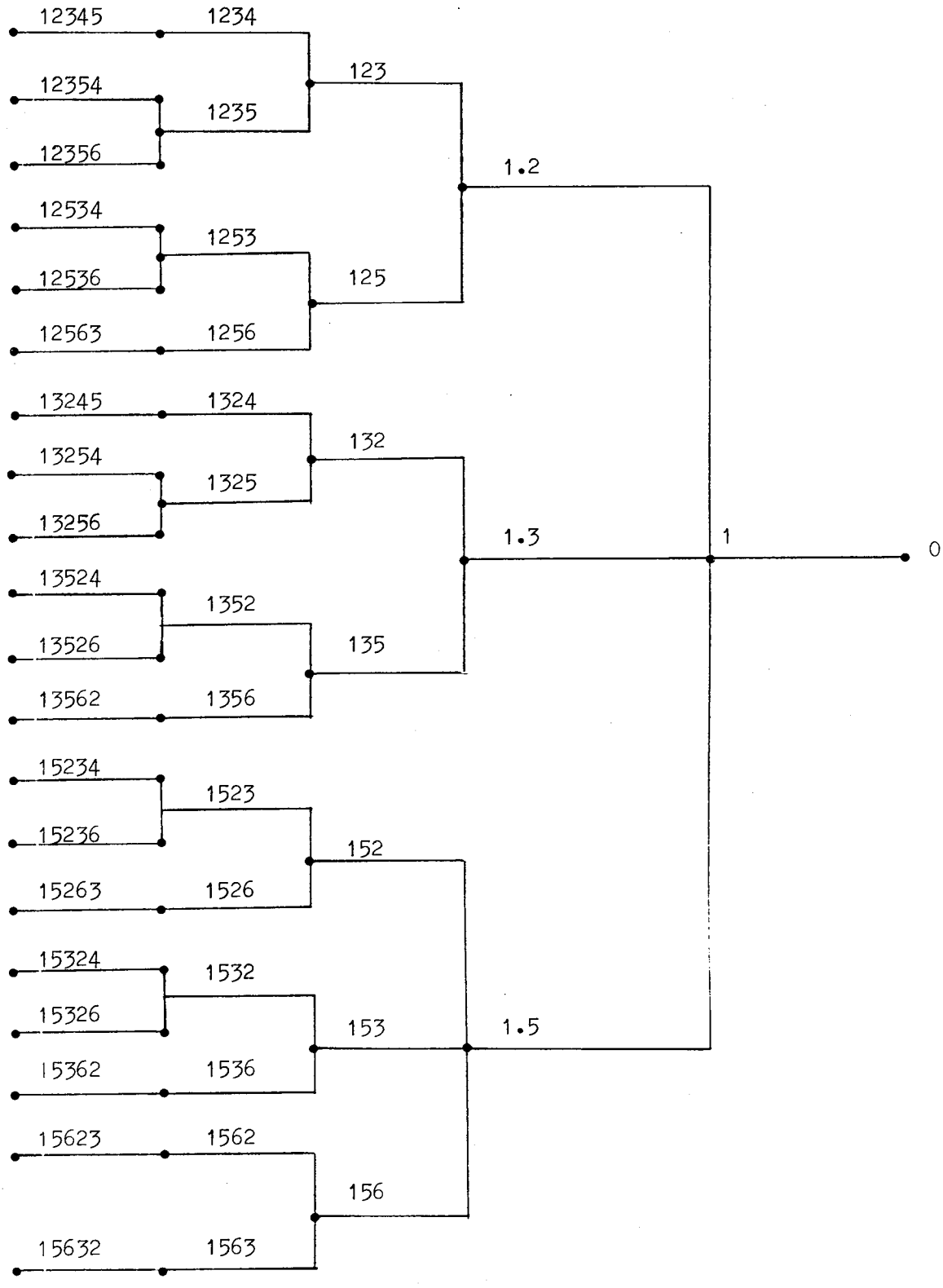
3 et 6 1.5.2

2 et 6 1.5.3

2 et 3 1.5.6

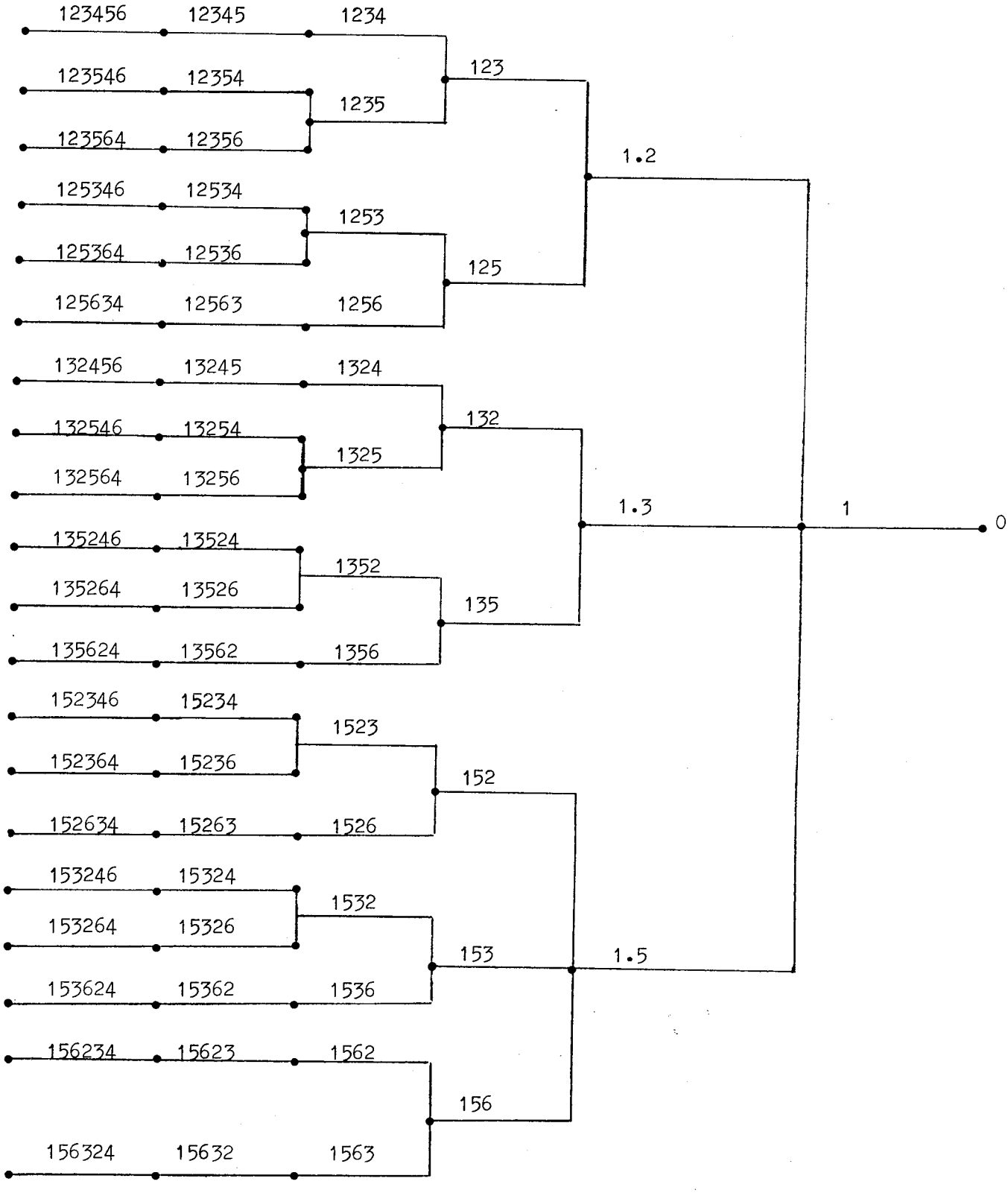


6) Seul (s)	5	a (ont) tous ses (leurs) prédécesseurs inclus dans	1234
	4 et 6		1235
	4 et 6		1253
	3		1256
	5		1324
	4 et 6		1325
	4 et 6		1352
	2		1356
	4 et 6		1523
	3		1526
	4 et 6		1532
	2		1536
	3		1562
	2		1563


12345
1234
12354
123
1235
12356
1.2
12534
1253
12536
125
12563
1256
13245
1324
132
13254
1325
13256
1.3
1
0
13524
1352
13526
135
13562
1356
15234
1523
15236
152
15263
1526
15324
1532
15326
153
1.5
15362
1536
15623
1562
156
15632
1563

7) Seul 6 a tous ses prédécesseurs inclus dans 12345

6	12354
4	12356
6	12534
4	12536
4	12563
6	13245
6	13254
4	13256
6	13524
4	13526
4	13562
6	15234
4	15236
4	15263
6	15324
4	15326
4	15362
4	15623
4	15632



L'arbre résiduel ne comporte dans cet exemple que 20 feuilles au lieu des 720 de l'arbre naturel.

IV. 3. 3. Une méthode séquentielle pour la génération séquentielle de l'arbre résiduel.

IV. 3. 3. 1. La méthode.

Initialisation :

On pose la racine 0. La permutation considérée est initialisée à $\{0\}$.
$y \leftarrow \{0\}$.

On dresse la liste des tâches sans prédécesseurs que l'on place dans $av(1)$.

On initialise le compteur de niveau à 1. i=1.

Niveau i :

Tant que $av(i) \neq 0$

- Choisir une tâche dans $av(i)$.
- L'ajouter à y.
- L'enlever de $av(i)$.
- Si i=n
  - Imprimer y.
- Sinon
  - Dresser la liste des tâches dont tous les prédécesseurs sont dans y et qui n'y sont pas elles-mêmes.
  - Placer cette liste dans $av(i+1)$.
  - Lancer la procédure récursive de génération au niveau i+1.
- Enlever la tâche choisie de y.

IV. 3. 3. 2. Illustrations.

Illustration 4. La génération séquentielle de l'arbre résiduel.

Reprenons l'exemple de IV. 3. 2. On a n=6.

$av(1) \leftarrow 1$ $\quad y = \{0\}$

Pas 1. Niveau 1 :

On choisit 1 dans $av(1)$.

On l'ajoute à y : $y = \{0,1\}$.

On l'enlève de av(1) av(1) = ∅.

i≠ n donc on dresse la liste, soit 2.3.5

av(2) = {2,3,5}.

On lance la génération au niveau 2.

Pas 2. Niveau 2 :

On choisit une tâche dans av(2) par exemple 2.

On l'ajoute à y : y = {0,1,2}.

On l'enlève de av(2) av(2) = {3,5}

i≠ n ⇒ av(3) = {3,5}.

Pas 3. Niveau 3 :

y = {0,1,2,3}

av(3) = {5}

av(4) = {4,5}.

Pas 4. Niveau 4 :

y = {0,1,2,3,4}

av(4) = {5}

av(5) = {5}.

Pas 5. Niveau 5 :

y = {0,1,2,3,4,5}

av(5) = ∅

av(6) = 6.

Pas 6. Niveau 6 :

y = {0,1,2,3,4,5,6}

On imprime y.

On enlève 6 de y y = {0,1,2,3,4,5}

av(6) =∅ donc fin.

Pas 7. Niveau 5 :

On enlève 5 de y y = {0,1,2,3,4}

av(5) = ∅ donc fin.

Pas 8. Niveau 4 :

On enlève 4 de y $y = \{0,1,2,3\}$

$av(4) \neq 0$ donc $y = \{0,1,2,3,5\}$

$av(4) = \emptyset$

$av(5) = \{4,6\}$.

Pas 9. Niveau 5 :

...

etc...

Illustration 5.

Texte de la prccédure Algol réalisant cet algorithme :

```
06450     'PROCEDURE' SCHEDULINGSTEPI;
06460     'BEGIN'
06530
06540     COMPUTEAVOFY;
06550     COMPUTESIGMAY;
06560     'IF' NODEADLINE 'THEN'
06570          'BEGIN' AVOFIISSETTOAVOFY;
06580                  ELIMINATION1;
06590                  'FOR' DUMMY:=0 'WHILE' LENGTHOFAV[I]>0 'DO'
06600                   'BEGIN'
06610                         Y[I]:=CHOISIR;
06620                         'IF' ¬ ELIMINATION2 'THEN'
06630                              'BEGIN'
06640                                   SCHEDULE(Y[I]);
06650                                   'IF' I=N 'THEN'
06660                                        IMPRIMER
06670                                   'ELSE'
06680                                        'BEGIN' I:=I+1;
06690                                                SCHEDULINGSTEPI;
06700                                                I:=I-1;
06710                                        'END';
06720                              'END' ELIM2=FALSE;
06730                         DISCARDFROMAVOFI(Y[I]);
06740                   'END' FOR EACH TASK OF AV[I];
06750          'END' IF NO DEADLINE OVER PASSED;
06760
06770     'END' SCHEDULING STEP I;
```

Les numéros de ligne renvoient au texte du programme donné en annexe 1.

IV. 4. GENERALISATION DE LA SHP

IV. 4. 1. Présentation.

Nous avons défini la SHP sur l'ensemble des permutations d'ordre n. Nous allons maintenant la généraliser à tous les nœuds de l'arbre naturel.

A un nœud est associée une sous-permutation.

A ce nœud nous allons associer le sous-planning obtenu en ordonnançant les tâches dont les numéros figurent dans la sous-permutation, dans l'ordre où elles y figurent, selon la SHP, comme si la permutation était complète, nous contentant de nous arrêter à la fin de la sous-permutation, ou quand les liens de précédence sont violés.

Comme nous avons vu comment générer l'arbre résiduel concordant avec le graphe de précédence, nous nous limiterons à ses nœuds et feuilles.

IV. 4. 2. Notations.

Nous noterons :

y : une sous-permutation,

y(0) : l'origine 0,

y(i) : le $i^{ème}$ élément de la sous-permutation y, i restant inférieur ou égal au nombre d'éléments de y.

$\Gamma = SHP(y,G,ha,\omega)$: Le planning associé à la sous-permutation y en accord avec les liens de précédence, les heures d'arrivée, les demandes de ressources.

$\Gamma = SHP(y)$: Par abus d'écriture, $SHP(y,G,ha,\omega)$ sera noté également $SHP(y)$ puisque G,ha,ω sont des constantes du problème.

av(i,y) : L'ensemble des éléments compléments intéressants de $\{y(0), y(1), \ldots y(i-1)\}$ c'est-à-dire dont tous les

prédécesseurs sont inclus dans $\{y(0), y(1), \ldots y(i-1)\}$ et n'y sont pas eux-mêmes. Cette constante du problème ne doit pas être confondue avec la variable av(i) du § 3.

$\sigma(t,i,y)$: L'heure de départ au plus tôt de la tâche t dans le planning $\Gamma = SHP(\{y(0), \ldots, y(i-1),t\})$.

lib(a,i,y) : L'heure à laquelle la ressource "a" se trouve libre quand on a ordonnancé $\{y(0), y(1), \ldots, y(i)\}$ dans cet ordre selon la SHP. C'est-à-dire l'heure à laquelle se termine la dernière de ces tâches placée sur "a".

IV. 5. UN ALGORITHME POUR L'ORDONNANCEMENT DE TACHES NE NECESSITANT CHACUNE QU'UNE SEULE RESSOURCE UNIQUE DANS SON TYPE [SHRAGE]

IV. 5. 1. Présentation.

SHRAGE démontre en [2] que s'il n'existe qu'une seule ressource par type, et que chaque tâche ne demande qu'une seule ressource une méthode qu'il ne définit que par ses propriétés (cf.IV.5.2.) permet de générer tous les plannings actifs possibles, et uniquement les plannings actifs grâce à certains critères d'élimination de nœuds complémentaires.

Il montre comment coupler l'algorithme de génération séquentielle de l'arbre résiduel, et l'algorithme de génération de plannings à partir de permutations, afin d'obtenir un algorithme de génération séquentielle de plannings actifs, sans régénérer les parties communes et sans générer de plannings inactif

Le principe en est le suivant : on utilise l'algorithme de HELLER-LOGEMANN pour générer un nœud et la permutation 'y' associée mais en plus d'ajouter un élément à 'y', on ajoute la tâche dans le planning associé ; et ce sont à la fois les propriétés de 'y' mises en évidence par HELLER-LOGEMANN, et celles du planning associé, mises en évidence par SHRAGE, qui vont servir à définir l'ensemble des successeurs intéressants ; ou le cas échéant abandonner un nœud.

IV. 5. 2. Définitions et notations.

La méthode de placement dans le planning n'est pas définie, SHRAGE se contente de dire que l'on calcule la date de départ au plus tôt, et que si le planning initial était actif, le planning complété l'est aussi. Si dans le cas réduit, il est relativement simple de trouver des méthodes de placement vérifiant ces hypothèses, (calculabilité de la date de départ, conservation de l'activité) ; dans le cas général traité ici, c'est beaucoup plus difficile à trouver, et surtout à démontrer. Pour illustrer la méthode de SHRAGE dans le cas réduit, nous prendrons une variante de la SHP avec récupération des trous. Nous ne démontrerons pas qu'elle vérifie les hypothèses, puisqu'un travail global est fait au chapitre V.

Les notations seront celles définies au § IV.4. par souci d'homogénéité.

On définit de plus une relation d'ordre sur les tâches, par exemple l'ordre naturel de leurs numéros, que l'on note $<$.
$y(i) < y(j)$ signifie que $y(i)$ est plus petite que $y(j)$ selon cette relation.

IV. 5. 3. Propriétés.

PROPRIETE N°1.

Considérons les ensembles $av(i,y)$. S'il existe dans $av(i,y)$ deux tâches t et t' telles que :

$$\sigma(t,i,y) + d(t) \leq \sigma(t',i,y)$$

alors, le nœud associé à la permutation $\{y(0), y(1), \ldots y(i-2), y(i-1), t'\}$ et tous ses descendants conduisent à des plannings inintéressants.

PROPRIETE N° 2.

Considérons les permutations y et les plannings associés. S'il existe un entier i, tel que :

$$\sigma(y(i),i,y) < \sigma(y(i-1),i-1,y)$$

ou

$$\begin{cases} \sigma(y(i),i,y) = \sigma(y(i-1),i-1,y) \\ \text{et} \\ y(i) < y(i-1) \end{cases}$$

alors, le nœud associé et tous ses descendants conduisent à des plannings inintéressants.

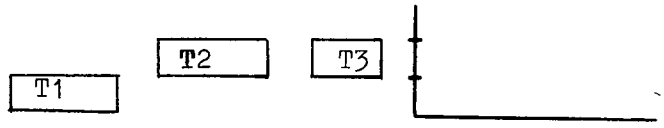
IV. 5. 4. La méthode.

- On utilise l'algorithme de HELLER-LOGEMANN.
- On initialise à chaque niveau de récursion la variable av(i) avec la constante av(i,y) comme précédemment.
- On examine cet ensemble av(i,y) et le planning associé à y pour voir si la propriété n° 1 de SHRAGE est vérifiée par certaines tâches, auquel cas on les enlève de av(i).

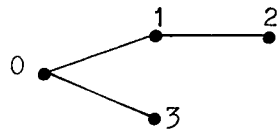
- Après chaque choix d'un nouvel élément pour y, on vérifie que la propriété n° 2 n'est pas vérifiée. Si elle l'est, on remonte au niveau supérieur.

IV. 5. 5. Exemple de récupération de trous.

Demande : deux types de ressources, une ressource par type (hypothèse de la méthode).

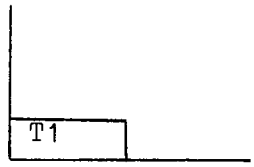


Précédence :

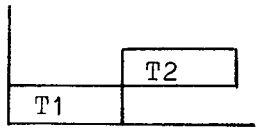


Pas 1. Etude de la permutation 1.2.3.

α) On place T1.

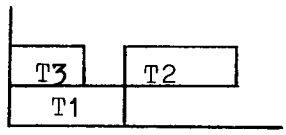


β) On place T2.



γ) On place T3.

La méthode de SHRAGE exige que l'on récupère les trous. On cherche donc où placer T3, on la place dans le trou :



mais alors la propriété n° 2 est vérifiée, on abandonne donc la permutation.

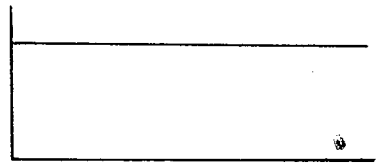
IV. 5. 6. La généralisation qui en est donnée par SHRAGE.

SHRAGE donne une méthode pour traiter le cas des demandes multiples telles que définies au § IV.1.1. (p. 80). On ne dit pas sur quelle machine les tâches sont placées. On se contente pour chaque type de diminuer les disponibilités de $\omega(t,j)$ durant la période où est placée la tâche t, et l'on récupère les trous en cherchant quand assez de ressources sont disponibles dans tout le planning ou plutôt dans ce qui en tient lieu, et non à la fin comme pour la SHP.

IV. 5. 7. Exemple d'utilisation de la généralisation de SHRAGE.

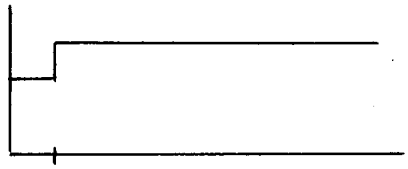
Reprenons l'exemple du § IV.1.3 (p. 81).

La permutation 1.2.3.4 est traitée ainsi :



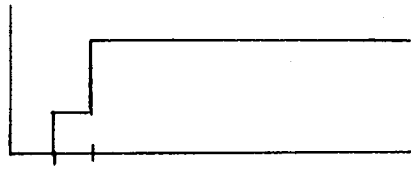
Limite : 3 Processeurs.

α) Tous les processeurs sont libres. On place T1 en 0.



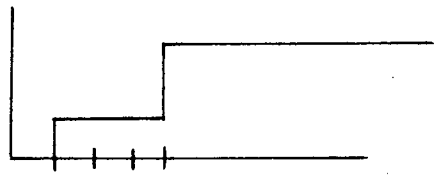
La limite est descendue en 0,durant une unité de temps,de une unité de ressource.

β) On place T2 en 0.

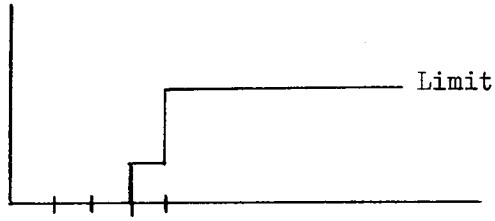


La limite est descendue entre 0 et 2 de deux unités de ressource.

γ) On place T3 en 2.



δ) On place T4 en 1.



Limite finale.

IV. 5. 8. Commentaires sur la méthode de SHRAGE généralisée.

Cette méthode oblige à avoir en mémoire tous les instants du planning, et à les considérer tous pour trouver la place d'une tâche. Ce travail est très long, et il faut le multiplier par le nombre de types de ressources au moins.

Le questionnaire étant de la forme :

Temps $\mathfrak{t} := 0$

Type 1 : Trouver $\mathfrak{t}'$ le premier instant suivant $\mathfrak{t}$ où il y ait :
$\omega(t,1)$ ressources libres. $\mathfrak{t} := \mathfrak{t}'$.
Ces ressources sont-elles libres sur le segment
$[\mathfrak{t}, \mathfrak{t} + d(t)]$? OUI passer au type 2 ;
NON boucler pour trouver une autre période de liberté.

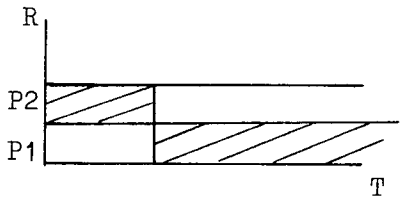
Type 2 : Trouver le premier instant $\mathfrak{t}'$ suivant $\mathfrak{t}$ où il y ait :
$\omega(t,2)$ ressources libres. $\mathfrak{t} := \mathfrak{t}'$.
Les ressources de type 1 sont-elles libres sur le segment
$[\mathfrak{t}, \mathfrak{t} + d(t)]$? NON repartir au type 1 ;
OUI :
Les ressources de type 2 sont-elles libres jusqu'à :
$\mathfrak{t} + d(t)$? OUI passer au type 3 ;
NON boucler pour trouver une autre période de liberté.

et ainsi de suite pour tous les types.

Si la calculabilité d'une date de départ est évidente, la conservation de l'activité l'est moins et la faisabilité du planning ainsi calculé n'est pas démontrée. En particulier, sur ces bases, SHRAGE généralise son algorithme au cas non résolu où les ressources ne sont pas toujours disponibles.

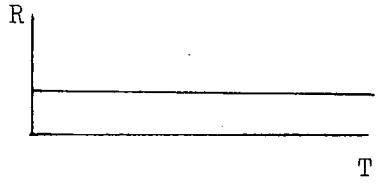
La généralisation étant faite comme suit, la limite est dès l'origine non linéaire et l'on travaille comme précédemment (§ IV.5.7). Hélas les plannings

obtenus ne sont pas faisables :



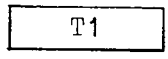
P2 indisponible avant T=3.

P1 indisponible après T=3.



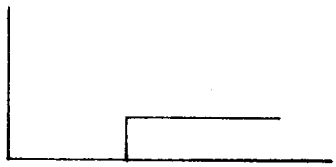
Limite initiale : droite y=1.

Demande :

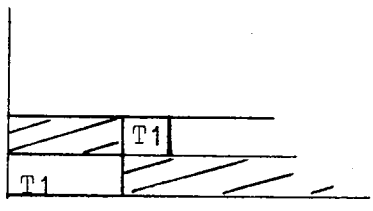


Résolution :

On place T1 en 0 dans l'ersatz de planning :



Limite après placement de T1. Le problème semble faisable.



Le seul planning répondant aux heures d'activation trouvées ne vérifie pas la non-préemption.

Il est évident que ce contre-exemple est simpliste, mais il en existe d'autres moins simples et beaucoup plus courants. Son but n'est pas de contredire une affirmation que SHRAGE donne sans aucune garantie et sans y attacher d'importance, et qui reste valable dans certains cas (toute ressource une fois mise en service y reste) mais de montrer combien il est difficile de démontrer qu'une méthode de planification vérifie les hypothèses de SHRAGE. C'est pourquoi d'ailleurs, les démonstrations pour la SHP seront faites directement.

IV. 6. ORDON

IV. 6. 1. Présentation.

Les commentaires que nous venons de faire sur la méthode de SHRAG généralisée, nous conduisent à chercher une autre méthode, la SHP. Cette mé nous permet de ne mémoriser que la frontière du planning, et rien de ce qui passe avant, et la place d'une tâche est entièrement déterminée.

Nous allons montrer au chapitre V que cette méthode est équivale à celle de SHRAGE pour le problème de base et les problèmes qu'il a traités généralisation [2] que nous prenons nous comme problèmes de base.

Par équivalente, nous entendons que les mêmes solutions sont obte ou éliminées, et au même point de la génération. Celà tient au fait que si SHRAGE récupère les trous, ou du moins prévoit son algorithme pour les récu rer, nous démontrons qu'en utilisant la SHP, il n'y en a pas. Ceci constitu une démonstration indirecte du fait que dans la méthode de SHRAGE il est in tile de considérer les points placés avant la frontière.

Par problème de base, nous entendons celui défini par SHRAGE : un ressource par type, une ressource par tâche.

Par généralisations, nous entendons celles définies par SHRAGE : x ressources par type :

$\omega(i,j)$ ressources de type j pour la tâche i,
pour tous les types, toutes les tâches.

Nous étudierons ensuite deux autres généralisations, les relâches décalées, et les demandes sans contraintes (cf.Chapitre V § 5 et 6).

En annexe 2 sont donnés deux résultats récents.

IV. 6. 2. Méthode.

Elle est semblable à celle de SHRAGE, à ceci près, qu'au lieu d'utiliser une méthode définie par ses seules propriétés, comme il le fait pour les démonstrations, ou bien une méthode de recherche systématique dans tout le planning pour laquelle, de plus, bien que ce soit vrai, il n'est pas démontré qu'elle vérifie les propriétés nécessaires aux démonstrations ; nous utilisons la SHP généralisée, qui permet de ne considérer que la frontière, et nous montrerons au chapitre V qu'elle est optimale ce, de façon directe.

IV. 6. 3. Notations.

Rappelons qu'outre celles données au § 4 nous utiliserons une variable av(i) qui servira à mémoriser à chaque niveau de récursion les éléments de av(i,y) qu'il est utile d'examiner et qui ne l'ont pas encore été. Cette variable déjà utilisée au § 4 ne doit pas être confondue avec les constantes bien définies et fixes av(i,y).

IV. 6. 4. Illustration.

Illustration 6. L'algorithme ORDON.

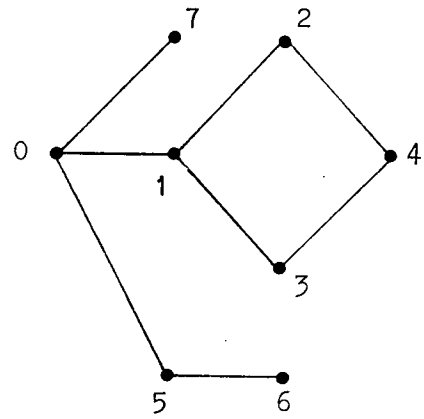
Soit le problème suivant (*) :

2 types de ressources M,P ;

2 ressources de chaque type P1,P2, M1,M2 ;

7 travaux ou tâches T1, T2, T3, T4, T5, T6, T7 ;

le graphe G :



les caractéristiques :

Tâche	ω(i,P)	ω(i,M)	D(i)	ha(i)
1	1	2	3	0
2	1	1	2	0
3	1	2	2	0
4	2	2	5	0
5	1	0	1	0
6	1	0	1	0
7	1	0	4	0

(*) Ce problème est un paradigme pour Ordon ; déjà utilisé en partie pour les illustrations précédentes, il sera désormais utilisé comme exemple dans la suite de ce travail ; en particulier, pour le programme ALGOL-60 (cf. Annexe 1).

Pas 1. Niveau 1 :

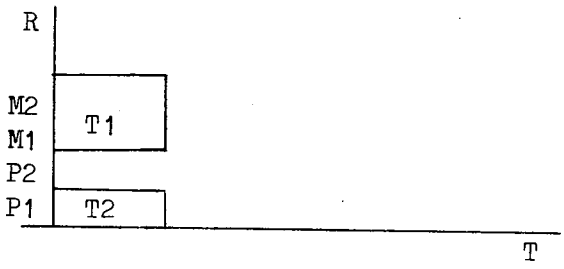
$y = \{0\}$, $lib(P1,0,y) = lib(P2,0,y) = lib(M1,0,y) = lib(M2,0,y) = 0$

$av(1) = av(1,y) = \{1,5,7\}$

$\sigma(1,1,y) = 0$; $\sigma(5,1,y) = 0$; $\sigma(7,1,y) = 0$.

On choisit T1, on lui alloue P1, M1, M2. $y = \{0,1\}$, $av(1) = \{5,7\}$.

$\Gamma(y) =$



Pas 2. Niveau 2 :

$y = \{0,1\}$, $lib(P1,1,y) = lib(M1,1,y) = lib(M2,1,y) = 3$

$lib(P2,1,y) = 0$

$av(2) = av(2,y) = \{2,3,5,7\}$

$\sigma(2,2,y) = 3$; $\sigma(3,2,y) = 3$; $\sigma(5,2,y) = 0$; $\sigma(7,2,y) = 0$

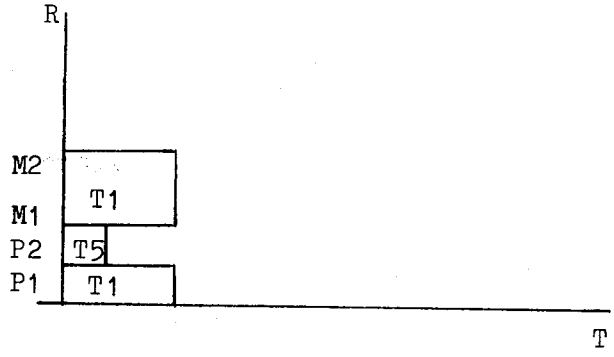
$\sigma(5,2,y) + d(5) < \sigma(2,2,y) \Rightarrow$ T2 éliminée de $av(2)$

$\sigma(5,2,y) + d(5) < \sigma(3,2,y) \Rightarrow$ T3 éliminée de $av(2)$

$av(2) = \{5,7\}$.

On choisit T5, on lui alloue P2. $y = \{0,1,5\}$, $av(2) = \{7\}$.

$\Gamma(y) =$



Pas 3. Niveau 3 :

$y = \{0,1,5\}$, $lib(P1,2,y) = lib(M1,2,y) = lib(M2,2,y) = 3$

$lib(P2,2,y) = 1$

$av(3) = av(3,y) = \{2,3,6,7\}$

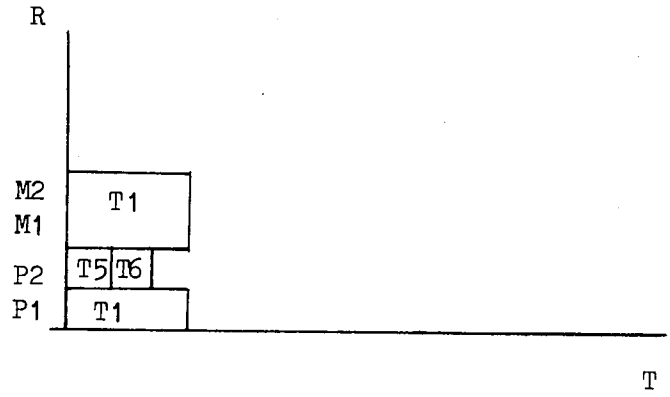
$\sigma(2,3,y) = 3$; $\sigma(3,3,y) = 3$; $\sigma(6,3,y) = 1$; $\sigma(7,3,y) = 1$

$$\left.\begin{array}{l} \sigma(6,3,y) + d(6) < \sigma(2,3,y) \\ \\ \sigma(6,3,y) + d(6) < \sigma(3,3,y) \end{array}\right\} \quad \text{T2, T3 éliminées de } av(3)$$

$av(3) = \{6,7\}$.

On choisit T6, on lui alloue P2. $y = \{0,1,5,6\}$, $av(3) = \{7\}$.

$\Gamma(y) =$



Pas 4. Niveau 4 :

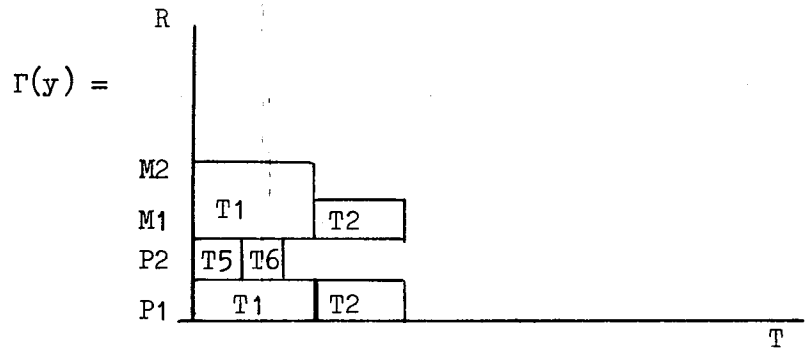
$y = \{0,1,5,6\}$, $lib(P1,3,y) = lib(M1,3,y) = lib(M2,3,y) = 3$

$lib(P2,3,y) = 2$

$av(4) = av(4,y) = \{2,3,7\}$

$\sigma(2,4,y) = 3$; $\sigma(3,4,y) = 3$; $\sigma(7,4,y) = 2$.

On choisit T2, on lui alloue P1, M1 (SHP). $y = \{0,1,5,6,2\}$, $av(4) = \{3,7\}$.



Pas 5. Niveau 5 :

$y = \{0,1,5,6,2\}$, $lib(P1,4,y) = lib(M1,4,y) = 5$,

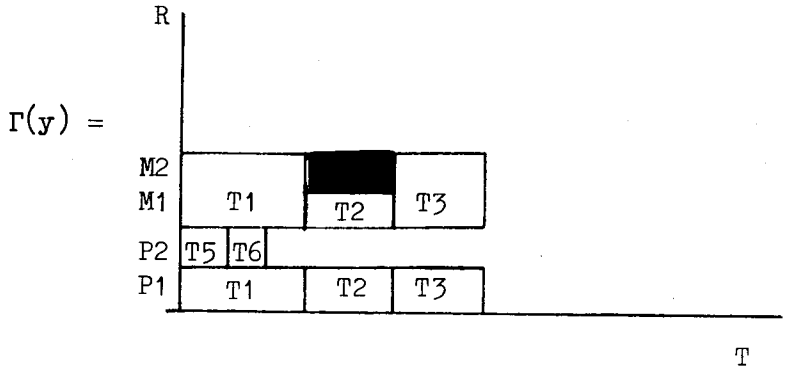
$lib(P2,4,y) = 2$, $lib(M2,4,y) = 3$

$av(5) = av(5,y) = \{3,7\}$

$\sigma(3,5,y) = 5$; $\sigma(7,5,y) = 2$.

On choisit T3, on lui alloue P1, M1, M2. $y = \{0,1,5,6,2,3\}$,

$av(5) = \{7\}$.



Pas 6. Niveau 6 :

$y = \{0,1,5,6,2,3\}$, $lib(P1,5,y) = lib(M1,5,y) = lib(M2,5,y) = 7$,

$lib(P2,5,y) = 2$

$av(6) = av(6,y) = \{4,7\}$

$\sigma(4,6,y) = 7$; $\sigma(7,6,y) = 2$

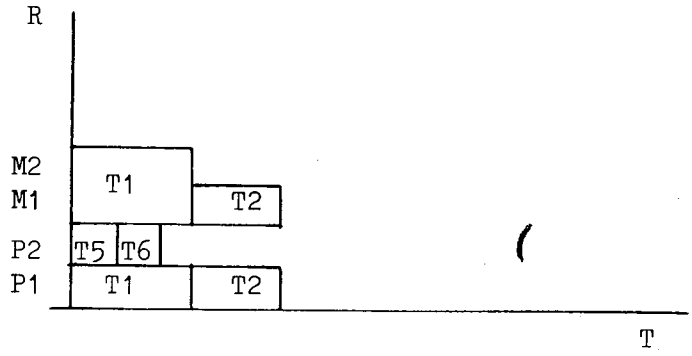
$\sigma(7,6,y) + d(7) < \sigma(4,6,y) \Rightarrow$ T4 éliminée de $av(6)$

$av(6) = \{7\}$.

On choisit T7. $y = \{0,1,5,6,2,3,7\}$ mais $\sigma(y(6),6,y) < \sigma(y(5),5,y)$ donc retour en arrière, on enlève T7 et T3 de y, on repart avec les données du niveau 5 là où elles en étaient restées.

$y = \{0,1,5,6,2\}$, $av(5) = \{7\}$.

$\Gamma(y) =$



Pas 7. Niveau 5 :

$y = \{0,1,5,6,2\}$, $lib(P1,4,y) = lib(M1,4,y) = 5$,

$lib(P2,4,y) = 2$, $lib(M2,4,y) = 3$

$av(5) = \{7\}$

$\sigma(7,5,y) = 2$.

On choisit T7. $y = \{0,1,5,6,2,7\}$ mais $\sigma(y(5),5,y) < \sigma(y(4),4,y)$ donc retour en arrière, on enlève T7 et T2, on repart au niveau 4.

Pas 8. Niveau 4 :

$y = \{0,1,5,6\}$, $lib(P1,3,y) = lib(M1,3,y) = lib(M2,3,y) = 3$,

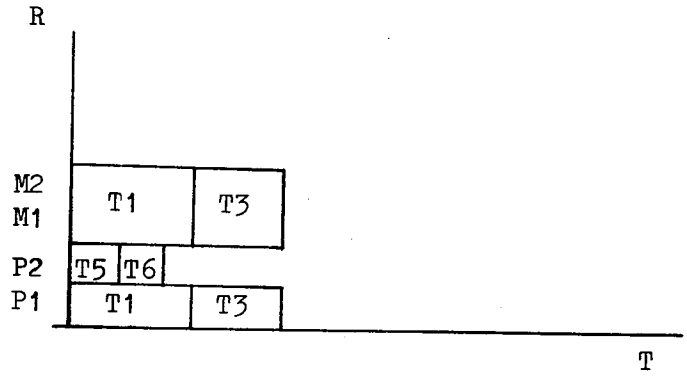
$lib(P2,3,y) = 2$

$av(4) = \{3,7\}$

$\sigma(3,4,y) = 3$; $\sigma(7,4,y) = 2$.

On choisit T3, on lui alloue T1, M1, M2. $y = \{0,1,5,6,3\}$,

$av(4) = \{7\}$.



... ...

REFERENCES

[1] HELLER J. LOGEMANN G. "An algorithm for the construction and evaluation of feasible schedules". Management Science, Vol.8, n° 2, pp.168-183. (1962).

[2] SHRAGE L. "Solving resource constrained network problems by implicit enumeration. Non preemptive case". Opns. Research. Vol.18, pp.263-278. (1970).

[3] RAND CORPORATION.

DREZNER S.M. VAN HORN R.L. "Design considerations for CAMCOS. A computer assisted maintenance planning and control system". Rand Memorandum RM 5255 PR. (July 1967).

GEOFFRION A.M. "Integer programming by implicit enumeration and Balas method". Rand Memorandum RM 4783 PR. (February 1966).

GEOFFRION A.M. "Implicit enumeration using an imbedded linear program". Rand Memorandum RM 5406 PR. (September 1967).

GEOFFRION A.M. "An improved implicit enumeration approach for integer programming". Rand Memorandum RM 5644 PR. (June 1968).

PRITSKER A.A.B. WATTERS L.J. "A 0/1 programming approach to scheduling with limited ressources". Rand Memorandum RM 5561 PR. (January 1968).

Chapitre cinquième.

Démonstrations.

TABLE DES MATIERES

LISTE DES DEFINITIONS, LEMMES, PROPRIETES, THEOREMES

Théorème 3 Surjectivité de la SHP.

Théorème 4 Comment utiliser les temps morts ? : pas 5.

V. 0. NOTATIONS : TABLEAU RESUME

n ..... : Nombre total de tâches.

r ..... : Nombre de types de ressources.

t ..... : Une tâche.

ha(t) ..... : L'heure d'arrivée de la tâche t, avant laquelle elle n'existe pas.

hdl(t) ..... : La date critique de la tâche t.

d(t) ..... : La durée de la tâche t.

$\omega(t,j)$ ..... : Le nombre de machines de type j que désire la tâche t.

Γ ..... : Un planning, qui peut n'être que partiellement défini.

y ..... : Une permutation des n premiers entiers, qui peut n'être que partiellement définie. En bijection avec un nœud ou une feuille de l'arbre.

y(i) ..... : Le $i^{ème}$ élément de la permutation y. Par abus d'écriture la tâche portant ce numéro.

pos(t,y) ..... : La position de la tâche t dans la permutation y ; i.e. l'entier i tel que y(i) = t.

$\sigma(t,i,y)$ ..... : L'heure de départ au plus tôt de la tâche t, si les i-1 premiers éléments de y sont ordonnancés selon la SHP et

dans cet ordre, et si t est placée en $i^{ème}$ position. Par abus d'écriture, l'heure d'exécution de t si elle est effectivement le $i^{ème}$ élément de y.

lib(a,i,y) ... : L'heure à laquelle la ressource 'a' est libérée après l'exécution des i premiers éléments de y placés dans cet ordre et selon la SHP.

av(i,y) ..... : L'ensemble des tâches dont tous les prédécesseurs sont compris dans les i-1 premiers éléments de y et qui ne le sont pas elles-mêmes. C'est-à-dire, les successeurs du nœud de l'arbre naturel réduit associé à la sous-permutatic y(1) ... y(i-1).

av(i) ..... : De la façon dont travaille l'algorithme un seul nœud y est considéré à la fois. C'est pourquoi on conserve dans une variable av(i) les éléments de av(i,y) (y étant le nœud courant) intéressants et non encore considérés.

V. 1. DEUX PROPRIETES ELEMENTAIRES

THEOREME 0.

SI à un pas i de la génération d'un planning Γ, il existe dans av(i,y) une tâche t, telle que $\sigma(t,i,y) > hdl(t) - d(t)$, c'est-à-dire que son heure d'activation au plus tôt dans Γ soit supérieure à son heure de départ au plus tard, alors il est inutile de poursuivre la génération du sous-arbre issu de y.

Démonstration.

Evidente, il est évident que t doit être placée dans Γ avant le pas i, afin d'être exécutée à temps.

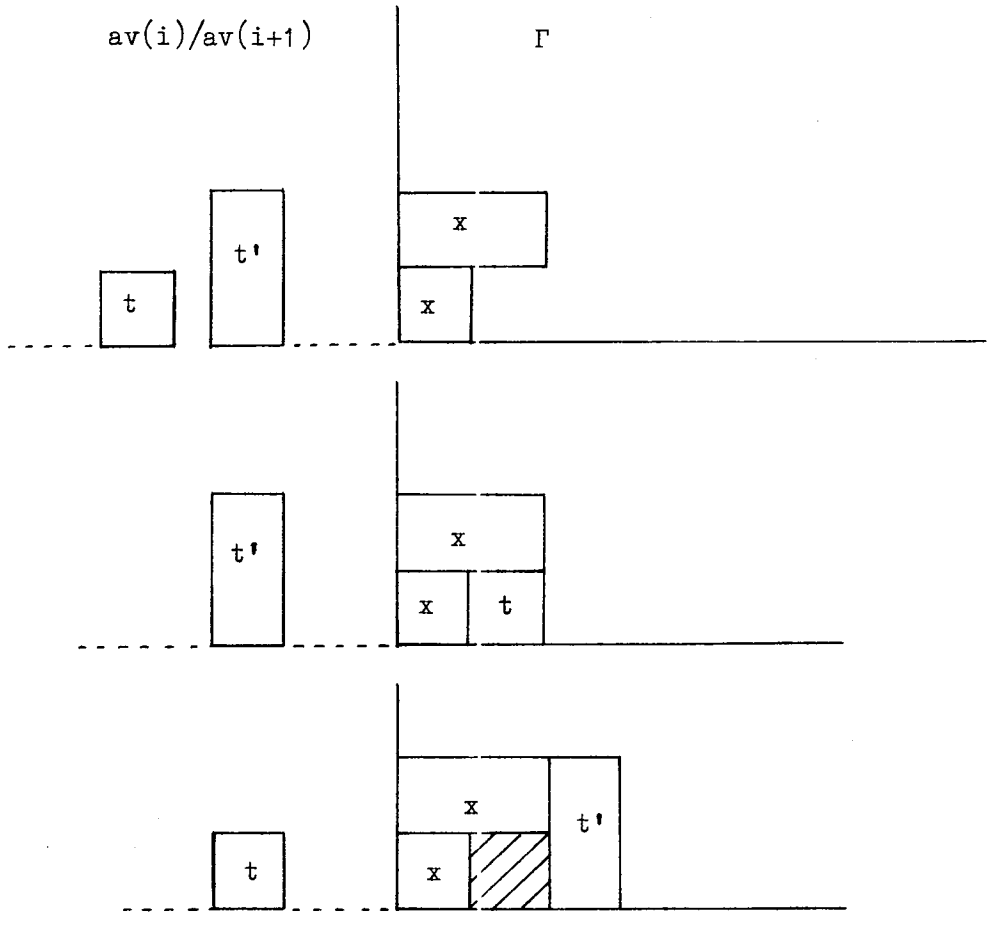
THEOREME 00.

Si à un pas i de la génération d'un planning Γ, il existe dans av(i,y) deux tâches, t et t', telles que $\sigma(t,i,y) + d(t) \leq \sigma(t',i,y)$ alors, t' peut être éliminée de av(i).

Démonstration.

Cette démonstration, donnée en [1] reste valable pour le problème étendu ; av(i) est l'ensemble des tâches qu'il peut être intéressant de plac dans Γ au pas i, t peut être placée au pas i, sans que t' ne soit retardée, tandis que placer t' au pas i soit ne changerait rien pour t, soit créerait un temps mort préjudiciable à t. Il est donc inutile d'essayer de placer t' au pas i.

Illustration.



Au pas i on peut placer t ou t'.

On place t, t' n'e pas retardée.

On place t', t est retardée par l'int duction d'un temps mort.

V. 2. UNE PREMIERE JUSTIFICATION DE LA SHP

Nous formulons ici quelques considérations qui permettent de justifie à priori le choix de la SHP. Une véritable preuve de sa validité sera donnée à posteriori, quand il sera démontré qu'il n'existe pas de meilleur ordonnancement que ceux trouvés par Ordon.

Si l'on considère comme admis le principe de base qui dit que l'on peut, en considérant toutes les permutations, considérer tous les plannings en procédant comme suit :

Les i-1 premières tâches étant ordonnancées, ordonnancer la $i^{ème}$ de sorte qu'elle parte le plus tôt possible après la frontière.

On constate qu'à un pas i, aucune tâche n'est retardée, si au lieu de choisir au hasard les ressources allouées à la tâche ordonnancée, on les choisit selon la SHP. (cf. LEMME N° 1).

Rappelons que cette étude a pour but de démontrer que l'image donnée par la SHP d'un sous-ensemble de Sn défini par certaines propriétés, contient tous les meilleurs ordonnancements, et, dans le cadre du temps réel, que les meilleurs :

- L'intuition de l'image de Sn et des propriétés est fournie par SHRAGE.
- L'intuition de la SHP est justifiée par le lemme n° 1.
- Les théorèmes qui suivent apporteront les démonstrations; en particulier le principe de base ne sera plus posé en axiome comme pour le lemme n° 1, mais démontré.

LEMME N° 1.

Si l'on considère que le principe de faire correspondre à toute permutation (ou sous-permutation) y un planning (ou sous-planning) Γ ; en plaçant successivement les tâches après la frontière dans l'ordre donné par y ,

de façon à ce qu'elles partent le plus tôt possible, peut fournir les meilleur ordonnancements; alors, il n'existe pas de meilleure politique d'attribution d ressources que la SHP.

Démonstration.

Nous allons démontrer qu'aucune tâche t' n'est retardée, si au pas une tâche t est placée selon la SHP, au lieu de l'être selon toute autre poli tique, sachant que t doit dans les deux cas être placée le plus tôt possible, en tenant compte des i- pas précédents.

1) Toute tâche t' déjà placée dans Γ n'est aucunement influencée pa le mode d'attribution des ressources de t, puisque le principe de l'algorithm est de placer t dans Γ sans modifier quoi que ce soit pour les tâches qui y so déjà placées.

2) La tâche t elle-même n'est nullement retardée, puisque la SHP implique de lui attribuer des ressources qui lui permettent de débuter à son heure de départ au plus tôt : $\sigma(t,i,y)$. Or, $\sigma(t,i,y) = h$ implique qu'à l'instant h, il y a au minimum assez de ressources libres pour satisfaire la deman de t. Ce qu'exige la SHP, c'est, s'il y a plus de ressources que nécessaire, de choisir parmi celles-ci, celles libérées le plus tard avant h, pas de reta der h.

3) Pour toute tâche t', placée après t dans la permutation y, trois cas se présentent :

3.1 : t doit précéder t', (contrainte de précédence) l'heure d'acti tion (donc de fin) de t n'ayant pas varié, t' n'est en rien influer

3.2 : t' est en concurrence avec t pour l'attribution de ressources Deux cas sont à considérer :

3.2.1 : t et t' sont en définitive exécutées sur des ressources dis tinctes. Là encore deux cas :

- 3.2.1.1/ Les ressources attribuées à t' sont toutes libérées avar

celles attribuées à t ; auquel cas, t' ne peut être retardée en s'exécutant sur les ressources qui lui sont attribuées.

- 3.2.1.2/ Il existe une ressource (r) attribuée à t', dont l'heure de libération est supérieure à l'heure de libération d'une de celles, (r_1 ... r_n), équivalentes, attribuées à t. Mais donc aussi à l'heure de départ de t ; sinon c'est cette ressource (r) et non une autre qui serait attribuée à t. Ainsi, faire exécuter t sur cette ressource (r) conduirait à retarder t ce qui est contraire au principe de base.

3.2.2 : t et t' sont exécutées sur au moins une même ressource. Celà implique que t' est exécutée après t, mais aussi que jusqu'à la fin de t, soit il n'existe pas de ressource du même type qui soit libre, soit t' ne peut être activée pour une autre raison. Dans le premier cas placer t sur une autre ressource la retarderait ; dans le second cela n'avancerait t' en rien.

V. 3. UN PREMIER ENSEMBLE DE THEOREMES

Après avoir ainsi justifié l'intuition de la SHP, nous allons poursuivre par des démonstrations ne nécessitant plus d'acte de foi.

LEMME N° 2.

Une fois que k-2 éléments ont été placés dans le sous-planning Γ, et sont devenus les k-2 premiers éléments de la sous-permutation y considérée. Si le $k^{ème}$ élément de y, soit t, peut débuter dans Γ avant le $k-1^{ème}$, soit t', c'est-à-dire :

$$\sigma(t,k,y) < \sigma(t',k-1,y)$$

alors il est inutile de considérer la sous-permutation y et ses descendants, puisqu'elle fournit le même sous-planning Γ que la sous permutation y' définie comme suit :

les k-2 premiers éléments sont inchangés	(on
t devient le $k-1^{ème}$ élément	(intervertit
t' devient le $k^{ème}$ élément	(t et t').

Démonstration.

1) $\sigma(t,k,y) < \sigma(t',k-1,y) \Rightarrow$ t et t' sont exécutées sur des ressource distinctes.

En effet, si ce n'était pas le cas, il existerait une ressource (au moins) sur laquelle t' serait exécutée avant t puisque précédant t dans la sous-permutation , et puisque toute tâche est placée derrière la frontière, sans récupération des trous ; et sur laquelle t serait exécutée avant t' par hypothèse. Comme l'inégalité est stricte, cela est impossible.

Donc :

2) $\sigma(t,k,y) < \sigma(t',k-1,y)$ implique qu'il existe assez de ressources pour exécuter t et t' en même temps. (Puisqu'elles sont exécutées sur des ressources distinctes).

3) Les heures d'activation de t et de t' seront inchangées que l'on considère y ou y'.

3.1 : $\sigma(t,k,y) = \sigma(t,k-1,y')$

En effet :

$\sigma(t,k,y) < \sigma(t,k-1,y')$ est impossible, puisque toutes choses restant égales par ailleurs, placer t' avant t dans Γ ne peut que retarder t ou ne rien changer.

$\sigma(t,k,y) > \sigma(t,k-1,y')$ est impossible.

Puisque celà impliquerait que, quand on a considéré y, t' ait reçu une ressource (au moins) qui aurait permis à t de s'exécuter plus tôt. Soit a cette (une de ces) ressource (s) et b celle allouée à t en échange.

$lib(a,k-2,y) < lib(b,k-1,y)$ puisque placer t sur b la retarde par

rapport à un placement sur a.

$lib(b,k-1,y) = lib(b,k-2,y)$ puisque b n'est pas utilisée par t'.

$lib(b,k-1,y) \leq \sigma(t,k,y')$ puisque t est exécutée sur b.

$\sigma(t,k,y) < \sigma(t',k-1,y)$ par hypothèse.

Donc :

$lib(a,k-2,y) < lib(b,k-1,y) = lib(b,k-2,y) \leq \sigma(t,k,y) < \sigma(t',k-1,y)$.

Ceci est en contradiction avec la SHP qui veut que dans ce cas b et non a soit allouée à t'.

3.2 : $\sigma(t',k-1,y) = \sigma(t',k,y')$

En effet :

$\sigma(t',k-1,y) > \sigma(t',k,y')$ est impossible, puisque toutes choses restant égales par ailleurs, placer t avant t' dans Γ ne peut que retarder t' ou ne rien changer.

$\sigma(t',k-1,y) < \sigma(t',k,y')$ est impossible puisque il existe assez de machines libres, à la date $\sigma(t',k-1,y)$ pour exécuter t et t'.

En effet :
quand on considère y, il existe pour tous les types de ressource j :

$\omega(t,j)$ machines x_{ij} tq $lib(x_i,k-1,y) \leq \sigma(t,k,y)$

$\omega(t',j)$ machines x'_{ij} tq $lib(x'_i,k-2,y) \leq \sigma(t',k-1,y)$

les x_{ij} étant allouées à t, x'_{ij} étant allouées à t'

et les deux ensembles sont disjoints (1) donc
$lib(x_i,k-1,y) = lib(x_i,k-2,y)$.

Donc au moins $\omega(t,j) + \omega(t',j)$ machines x''_{ij} tq
$lib(x''_{ij},k-2,y) < \sigma(t',k-1,y)$ puisque $\sigma(t,k,y) < \sigma(t',k-1,y)$.

Quand on considère y', t se verra octroyer $\omega(t,j)$ machines de type j prises parmi les x''_{ij}, et il en restera assez pour t'. Comme les relations de précédence n'ont pas varié (les k-2 premiers éléments sont invariants, t' peut partir à la même date.

4) Les mêmes ressources seront octroyées à t et t' dans les deux cas

4.1 : Pour les types de ressources où il n'y a pas concurrence, c'est évident.

4.2 : Pour le cas où il y a concurrence.
Pour une meilleure compréhension, nous traiterons d'abord le cas $\omega(t,j) = \omega(t',j) = 1$ et omettrons de préciser l'indice j. Néanmoins toutes les ressources considérées ici sont de type j. En 4.3 est donnée la démonstration avec double indice.

Dans le cas y (t' avant t) t' reçoit a, t reçoit b.

On en déduit :

$lib(a,k-2,y) \leq \sigma(t',k-1,y)$

$lib(b,k-2,y) = lib(b,k-1,y) \leq \sigma(t,k,y) < \sigma(t',k-1,y)$

d'où :

$lib(b,k-2,y) \leq lib(a,k-2,y)$ sinon b serait allouée à t'.

On en déduit de même :

$\nexists$ c tq $lib(b,k-2,y) < lib(c,k-2,y) \leq \sigma(t,k,y)$

$\nexists$ d tq $lib(a,k-2,y) < lib(d,k-2,y) \leq \sigma(t',k-1,y)$

c et d $\neq$ a et b.
Sinon c ou d serait allouée.

Dans le cas y' (t avant t').

Si $lib(a,k-2,y) < \sigma(t,k-1,y') = \sigma(t,k,y)$ alors t reçoit a. Mais alors t' reçoit b (ou une ressource e ayant la même date de libération que b) puisque :

$\nexists$ c $\nexists$ d $lib(b,k-2,y) < lib(c,k-2,y) \leq \sigma(t',k,y')$

$lib(b,k-2,y) < lib(d,k-2,y) \leq \sigma(t',k,y')$.

Si $lib(a,k-2,y) > \sigma(t,k-1,y')$ alors t reçoit b puisque $\nexists$ c, (ou bien une ressource e libérée à la même date), et alors t' reçoit a puisque $\nexists$ d, (ou bien une ressource f libérée à la même date que a).

Dans les deux cas, les deux mêmes ressources sont octroyées ou bien deux ressources ayant même date de libération.

4.3 : Considérons maintenant $\omega(t,j) = \beta_j$, $\omega(t',j) = \alpha_j$.

Dans le cas y, t' reçoit a_{ij}, $i = 1 \rightarrow \alpha_j$

t reçoit b_{ij}, $i = 1 \rightarrow \beta_j$

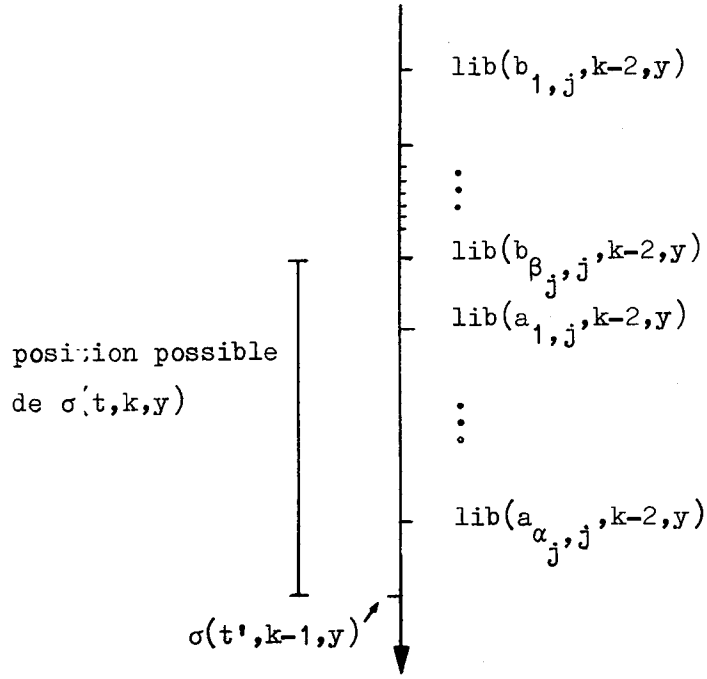
avec :

$lib(b_{\beta_j,j},k-2,y) < lib(a_{1,j},k-2,y)$

et $\neq c$, $\neq d$ ...

Dans le cas y', t reçoit tous les a_{ij} tq $lib(a_{ij},k-2,y) < \sigma(t,k-1,y'$ jusqu'à concurrence de β_j au maximum, et les $b_{i,j}$ nécessaires à compléter si besoin.t' reçoit les a_{ij} restants et les b_{ij} restants.

On constate que l'intervention d'une ressource e_j ou f_j ayant même date de libération et n'appartenant pas aux a_{ij} ou b_{ij} n'est possible que pour $b_{1,j}$, (dans tous les cas) et pour $a_{1,j}$ (uniquement si $lib(a_{1,j},k-2,y) > \sigma(t,k,y)$).



Donc dans les deux cas, le même ensemble de ressources (à 1 ou 2 près est alloué.

De plus dans les deux cas, $\omega(t,j)$ d'entre elles seront libérées à

$$\sigma(t,k,y) + d(t) = \sigma(t,k-1,y') + d(t)$$

et $\omega(t',j)$ seront libérées à :

$$\sigma(t',k-1,y) + d(t) = \sigma(t',k,y') + d(t)$$

les 1 ou 2 différentes étant libérées à : $lib(b_{1,j},k-2,y)$

et à : $lib(a_{1,j},k-2,y)$

dans les deux cas également (si différence il y a).

Conclusion.

Dans les deux cas, y ou y', les deux tâches t et t' sont exécutées à la même date, sur le même ensemble (en bloc) de ressources. Pour tous les types de ressources, on a, dans les deux cas, le même nombre de ressources libres à la même heure.

$\forall j$, $\exists$ n ressources $x_{i,j}$ tq $lib(x_{i,j},k,y) = h$

$\Rightarrow$ $\exists$ n ressources $x'_{i,j}$ tq $lib(x'_{i,j},k,y') = h$.

Donc y et y' mènent à deux plannings Γ et Γ' équivalents.

LEMME N° 3.

Une fois que k-2 éléments ont été placés dans le sous-planning Γ sont devenus les k-2 premiers éléments de la sous-permutation y considérée. Si le $k^{ème}$ élément de y, soit t peut débuter dans Γ à la même date que le $k-1^{ème}$, soit t', c'est-à-dire :

$$\sigma(t,k,y) = \sigma(t',k-1,y)$$

alors les sous-permutations y, précédemment définie, et y' définie comme su

les k-2 premiers éléments restent identiques (on intervertit
t devient le $k-1^{ème}$ (t
t' devient le $k^{ème}$ (et t').

conduisent au même sous-planning.

Démonstration.

Elle est identique à la précédente, en remplaçant les inégalités strictes par des inégalités non strictes, et en notant qu'il y a échange de ressources entre t et t'.

Conclusion.

Comme précédemment, on constate que les deux permutations y et y' mènent à deux sous-plannings équivalents, puisque il y a,après le pas k,le même nombre de ressources libres à la même date.

Les lemmes 2 et 3 conduisent au théorème 1 :

THEOREME 1.

Une fois que k-1 éléments ont été placés dans le sous-planning Γ et sont devenus les k-1 premiers éléments de la sous-permutation y considér (k-2 indifférents, t' étant le $k-1^{ème}$). Si l'on choisit pour être le $k^{ème}$ él une tâche t telle que :

$$\sigma(t,k,y) < \sigma(t',k-1,y)$$

ou

$$\sigma(t,k,y) = \sigma(t',k-1,y) \text{ et } t < t'$$

on ne fait que re-générer un sous-planning déjà examiné ou qui le sera. Il est donc inutile de poursuivre plus avant l'étude de y.

Démonstration.

Elle découle des lemmes 2 et 3. On a prouvé que y menait au même planning qu'une permutation y'.

Il faut donc choisir celle que l'on va étudier pour ne pas effectuer deux fois le même travail.

Dans le premier cas (inégalité), seule y possède une propriété distinctive. y' ne peut être détectée. Il est donc normal d'éliminer celle que l'on peut caractériser.

Dans le second cas, (égalité), la propriété est symétrique (y et y' la possèdent) on fait donc intervenir un critère de choix arbitraire, le plus simple possible, ici l'ordre lexicographique des noms de tâches.

Conclusion.

Elle constitue la propriété n° 1.

PROPRIETE N° 1.

Si l'on utilise l'algorithme de génération examen défini au § V, tout le sous-arbre issu de la sous-permutation y étant redondant avec celui issu de y'; on peut arrêter la génération à ce niveau.

LEMME N° 4.

Si l'on applique la propriété n° 1 à l'algorithme de génération, aucun des plannings Γ générés ne vérifie :

$$\exists\ k,l \text{ tq } \sigma(y(k),k,y) < \sigma(y(k-l),k-l,y)$$

(rappelons que y(k) est le $k^{ème}$ élément de la permutation y
(qui génère Γ)

i.e. les relations d'ordre : "place dans y" et "date de départ" sont compatibles pour les plannings générés.

Démonstration.

Si l'on utilise la propriété n° 1, tout planning généré vérifie $\sigma(y(k),k,y) \leq \sigma(y(k+1),k+1,y)$ ∀ k.

Donc :

$$\sigma(y(k-1),k-1,y) \leq \sigma(y(k-1+1),k-1+1,y)$$

$$\sigma(y(k-1+1),k-1+1,y) \leq \sigma(y(k-1+2),k-1+2,y)$$

... ...

... ...

$$\sigma(y(k-1),k-1,y) \leq \sigma(y(k),k,y)$$

⇒

$$\sigma(y(k-l),k-l,y) \leq \sigma(y(k),k,y)$$

∀ k ≤ nombre de tâches ∀ l ≤ k.

LEMME N° 5.

Si l'on applique la propriété n° 1 à l'algorithme de génération, aucun des plannings Γ générés ne vérifie :

$$\exists\ k,l \text{ tq } \begin{cases} \sigma(y(k),k,y) = \sigma(y(k-l),k-l,y) \\ \text{et } y(k) < y(k-l) \end{cases}$$

Démonstration.

Il ne peut exister l', $0 \leq l' \leq l$ tq

$$\sigma(y(k-l'),k-l',y) > \sigma(y(k),k,y) \text{ (Lemme 4)}$$

ou tq

$$\sigma(y(k-l),k-l,y) > \sigma(y(k-l'),k-l',y) \text{ (Lemme 4)}$$

Donc :

$$\sigma(y(k-l),k-l,y) = \sigma(y(k),k,y)$$

$$\Rightarrow \forall\ l',\ 0 \leq l' \leq l$$

$$\sigma(y(k-l),k-l,y) = \sigma(y(k-l'),k-l',y) = \sigma(y(k),k,y)$$

$$\text{donc } \forall\ l''\quad 0 \leq l'' < l$$

$$\sigma(y(k-l''-1),k-l''-1,y) = \sigma(y(k-l''),k-l'',y)$$

mais alors :

$$y(k-l''-1) < y(k-l'') \text{ (Propriété n° 1)}$$

donc :

$$y(k-l) < y(k-l+1)$$

$$y(k-l+1) < y(k-l+2)$$

... ...

... ...

$$y(k-1) < y(k)$$

$$\Rightarrow$$

$$y(k-l) < y(k)$$

Conclusion.

Les relations d'ordre total "place dans y" et "date de départ - ordre lexicographique" sont identiques pour les plannings générés.

Définition n° 1.

Nous définissons une application de l'ensemble des plannings dans l'ensemble des permutations de la façon suivante :

On classe les tâches incluses dans le planning selon leur date d'activation et, en cas d'égalité, selon l'ordre lexicographique de leurs noms. (Ou le cas échéant tout autre ordre choisi pour la propriété n° 1).

PROPRIETE N° 2.

La fonction classement ainsi définie est pour les plannings généré la réciproque de la fonction SHP.

Classement (SHP (y)) = y si SHP (y) existe.

Démonstration.

Si Classement (SHP (y)) = y' ≠ y

alors ∃ k,l tq $\sigma(y(k),k,y) < \sigma(y(k-1),k-1,y)$

ou tq $\begin{cases} \sigma(y(k),k,y) = \sigma(y(k-1),k-1,y) \\ \text{et } y(k) < y(k-1) \end{cases}$

puisque y(k-l) se retrouve après y(k) dans y' (deux permutations diffèrent s au moins deux éléments sont inversés y(k) après y(k-l) dans y et y(k) avant y(k-l) dans y').

Mais ceci est impossible (Lemmes 4 et 5) donc :

ou classement (SHP (y)) = y

ou $\nexists$ SHP(y)

THEOREME 2-0.

Une fois munie du critère d'arrêt donné par la propriété n° 1, la fonction SHP est injective :

i.e. : $\nexists$ y,y', y ≠ y', SHP(y) = SHP(y')

i.e. : il n'existe pas deux permutations menant au même planning, (même date d'activation, mêmes ressources pour toutes les tâches).

Démonstration.

Nous ne la ferons pas, puisque cette propriété est plus faible que celle du théorème 2.

THEOREME 2.

Une fois munie du critère d'arrêt donné par la propriété n° 1, la fonction SHP est telle qu'il n'existe pas deux permutations menant à deux plannings ayant même date d'activation pour toutes les tâches.

Démonstration.

S'il existe deux permutations y et y' menant à deux plannings équivalents.

Les dates d'activation et les noms de tâches étant identiques pour les deux plannings on a :

$$\text{Classement (SHP (y))} = \text{Classement (SHP (y'))}.$$

Or la propriété n° 2 implique :

$$\text{Classement (SHP (y))} = y$$

$$\text{Classement (SHP (y'))} = y'$$

donc $y = y'$.

LEMME N° 6.

Parmi les plannings générés, aucun ne contient de périodes d'inactivité assez importantes pour faire exécuter une tâche t quelconque plus tôt que prévu, durant l'une de ces périodes, sans retarder une autre tâche, ou faire exécuter un prédécesseur de t plus tôt.

Démonstration.

Pour ce premier lemme, nous ne voulons démontrer l'impossibilité de déplacer une tâche pour la mettre dans un "trou" existant sans violer les relations de précédence.

Le lemme 7 est quant à lui dédié au problème général.

Le "trou" peut être composé de plusieurs machines.

Considérons l'une des machines inactives.

S'il n'existe pas de tâche placée sur cette machine après la période d'inactivité, alors, la machine était disponible au moment de placer t, si t n'y a pas été placée, c'est que celà ne pouvait avancer sa date de départ.

S'il existe des tâches placées sur cette machine après la période d'inactivité considérée :

Soit u_1 la première de ces tâches.

Faisons de même avec toutes les machines incriminées (faisant parti du trou). On obtient pour chaque machine soit une première tâche u_i, soit rie s'il n'en existe pas.

Soit u la première des premières, c'est-à-dire la première de toute les tâches u_i, à avoir été placée dans le planning.

Rappelons la notation pos(x,y) pour position de la tâche x dans la permutation y et av(k,y) pour l'ensemble des tâches que l'on peut placer au pas k (précédence).

Deux cas sont possibles t appartenait à av(pos(u,y),y) (même si ell a été éliminée de av(pos(u,y)).). ou t n'appartenait pas à av(pos(u,y),y).

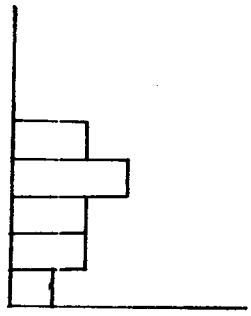
Si t appartenait à av(pos(u,y)) (même si elle a été éliminée de av(pos(u,y)).).

Dire qu'il existe un trou assez grand pour placer t cela signifie que $\sigma(t,pos(u,y),y)+d(t) < \sigma(u,pos(u,y),y)$ en effet u est placée de façon à être exécutée à la date $\sigma(u,pos(u,y),y)$. (Concept de base de la SHP). t peut être placée avant u, or au moment où l'on a placé u, les machines étaient lib

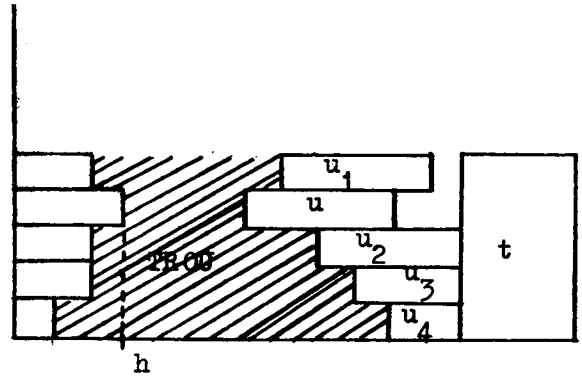
donc l'heure à laquelle aurait pu partir t **alors** ($\sigma(t,pos(u,y),y)$ est inférieu ou égale à celle où elle peut **partir maintenant** en bénéficiant du trou (h) (elle est en **fait égale**, nous **n'en** donnerons **pas** la démonstration) **et** comme le trou est **assez large** pour ne **pas** retarder u (hypothèse) :

$$\sigma(t,pos(u,y),y) \leq h$$

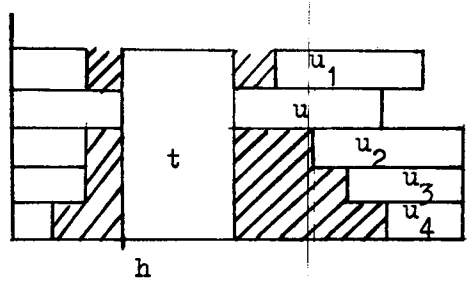
$$h+d(t) \leq \sigma(u,pos(u,y),y).$$



α) Etat au pas pos(u,y).



β) Etat final.



γ) Etat après déplacement de t.

Mais alors, le théorème 00 nous aurait fait éliminer u à ce pas.

Si t n'appartenait pas à av(pos(u,y),y), à cela, deux raisons possibles :

- t était déjà ordonnancée, auquel cas au moment où on a placé t, les machines étaient libres, donc si t avait gagné à y être placée elle y aurait été.

- Un prédécesseur de t n'était pas encore ordonnancé, ce prédécesseur a donc été ordonnancé après u et a donc une date de départ supérieure ou égale à celle de u (Lemmes 4 et 5) comme t ne peut débuter avant l'un de ses prédécesseurs, t ne peut débuter avant u si l'on n'avance pas ce prédécesseur.

Conclusion.

Soit le planning n'est pas généré à cause du théorème 00.

Soit t ne peut débuter plus tôt.

Soit il faut déplacer un des prédécesseurs de t.

Il est donc impossible de trouver un "trou" dans les plannings générés assez grand pour y placer une tâche de façon à avancer son exécution. Le lemme 7 va prouver quant à lui qu'aucun ré-arrangement ne peut permettre de faire partir une tâche plus tôt sans en retarder une autre. Nous aurons ainsi prouvé que les plannings générés sont actifs.

LEMME N° 7.

Il n'existe aucun moyen de ré-arranger un planning généré de façon à ce qu'une tâche t débute plus tôt que prévu, sans en retarder une autre. i.e. les plannings générés sont actifs.

Démonstration.

Considérons un planning Γ généré, $\Gamma = SHP(y)$ et représenté par :

$y = \{y(1), \ldots, y(n)\}$ la permutation d'origine ;

$\{\sigma(y(1),1,y), \ldots, \sigma(y(n),n,y)\}$ les heures de départ selon la SHP ;

$\{mach(y(1)), \ldots, mach(y(n))\}$ les ressources allouées selon la SHP.

Considérons le planning Γ' donné par le remaniement de Γ et représenté par :

$\{\sigma'(y(1)), \ldots, \sigma'(y(n))\}$ les nouvelles heures de départ ;

$\{mach'(y(1)), \ldots, mach'(y(n))\}$ les nouvelles ressources ;

$y' = \{y'(1), \ldots, y'(n)\}$ = Classement (Γ') le nouvel ordre de départ.

Soit i le plus petit entier tel que :

$$\sigma(y(i),i,y) \neq \sigma'(y(i))$$

ou ou et

$$mach(y(i)) \neq mach'(y(i))$$

1) $\sigma(y(i),i,y) \neq \sigma'(y(i))$

alors :

$\sigma(y(i),i,y) > \sigma'(y(i))$ sinon y(i) aurait été retardée.

Deux cas :

1.1 : Si $\forall k$, $k < i$, $pos(y(k),y') < pos(y(i),y')$.
C'est-à-dire si les tâches débutant avant y(i) dans Γ, débutent avant y(i) dans Γ'.

Rappelons que Γ et Γ' sont identiques pour ces tâches puisque i est le premier entier.

Il y a alors contradiction avec la SHP, puisque y(1) ... y(i-1) étant placées, y(i) est placée derrière de façon à débuter le plus tôt possible. Γ et Γ' étant jusqu'à présent identiques, le plus tôt de l'un est égal au plus tôt de l'autre donc y(i) ne peut partir plus tôt dans Γ' que dans Γ.

1.2 : Si il existe une tâche y(k), $k < i$, placée après y(i) dans Γ' $pos(y(i),y') < pos(y(k),y')$ alors qu'elle était placée avant y(i) dans $pos(y(i),y) = i > pos(y(k),y) = k$.

Comme toutes les tâches $y(1) \ldots y(k) \ldots y(i-1)$ sont placées de manière identique dans Γ et Γ' cela signifie que $y(i)$ a été placée dans un trou de Γ pour donner Γ'.

Le lemme 6 prouve que cela est impossible (Nota, si l'on avait dép un prédécesseur de $y(i)$ pour permettre ce transfert, i ne serait p le premier entier).

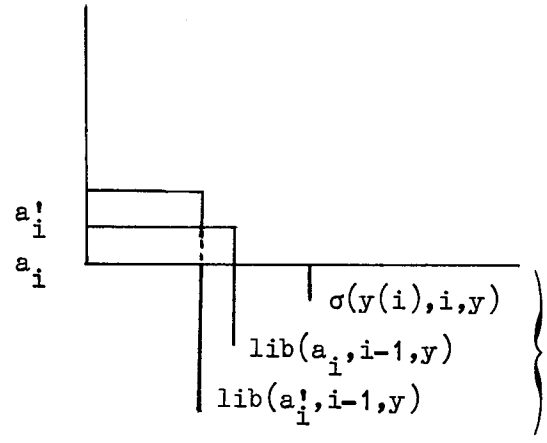
Donc $\sigma'(y(i),i,y) = \sigma'(y(i))$ pour i le premier entier...

2) $mach(y(i)) \neq mach'(y(i))$.

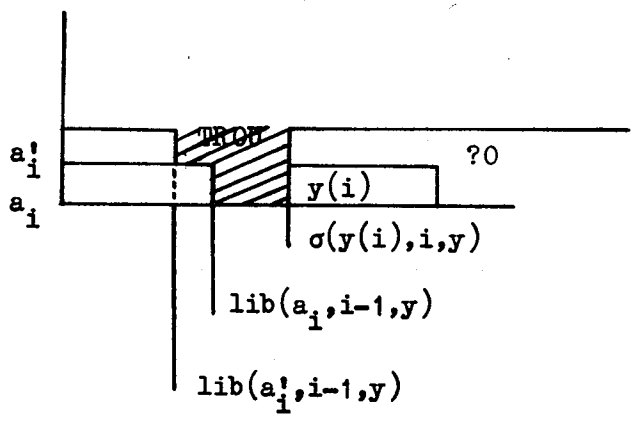
Considérons toutes les paires de ressources a_i, a'_i de même type, a_i étant allouée à $y(i)$ par la SHP, a'_i par la nouvelle méthode.

$lib(a_i,i-1,y) \geq lib(a'_i,i-1,y)$ car avant de placer $y(i)$, la date de libération de a_i est supérieure ou égale à celle de a'_i, ou alors la date de libération de a'_i est supérieure à $\sigma(y(i),i,y)$ (Définition de la SHP). Le sec cas est impossible (sinon placer $y(i)$ sur a'_i dans Γ' la retarderait).

Donc après avoir placé $y(1) \ldots y(i-1)$, le diagramme d'occupation des deux machines est (pour Γ comme pour Γ') :

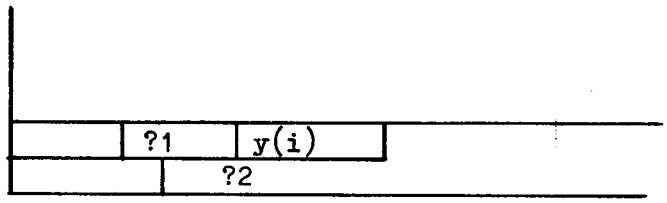


Γ ne peut avoir que la forme suivante :



puisque toute tâche placée après y(i) a une date de départ supérieure ou égal à celle de y(i).

Γ' a lui la forme suivante :



avec ?0, ?1, ?2 trois ensembles inconnus de tâches ou de périodes d'inactivité

Mais dire qu'il existe dans ?1 une tâche t conduit à envisager quatre cas.

2.1 : Cette tâche appartenait à av(i,y), auquel cas, elle peut avoir

2.1.1 : été placée dans Γ' avant y(i-1) donc dans un trou de Γ, puisque y(1) ... y(i-1) ont des positions identiques dans Γ et Γ' ce qui est impossible (Lemme 6) ;

2.1.2 : après y(i-1) et avant y(i) mais alors déjà à la génération

de Γ on aurait pu la placer ainsi, avec :

$\sigma(t,i,y) + d(t) \leq \sigma(y(i),i,y)$

ce qui est impossible (théorème 00).

2.2 : Cette tâche n'appartenait pas à av(i,y).

2.2.1 : Cette tâche n'appartenait pas à av(i,y) parce que déjà ordonnancée, $pos(t,y) < pos(y(i),y) = i$, auquel cas elle est placée dans Γ' comme dans Γ puisque i est le premier entier.

2.2.2 : Cette tâche n'appartenait pas à av(i,y) parce que l'un de ses prédécesseurs n'était pas ordonnancé. Dans ce cas, ce prédécesseur a été avancé dans Γ' par rapport à Γ. Considérons le premier des prédécesseurs déplacé. S'il a été placé avant y(i-1), il a été placé dans un trou, puisque y(1) ... y(i-1) ont des positions identiques dans Γ et Γ'. S'il a été placé après y(i-1) alors, soit t' son nom, on a :

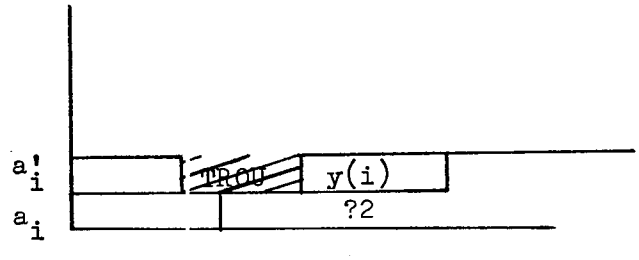
$\sigma(t',i,y) + d(t') \leq \sigma'(t)$

$\sigma'(t) + d(t) \leq \sigma'(y(i))$

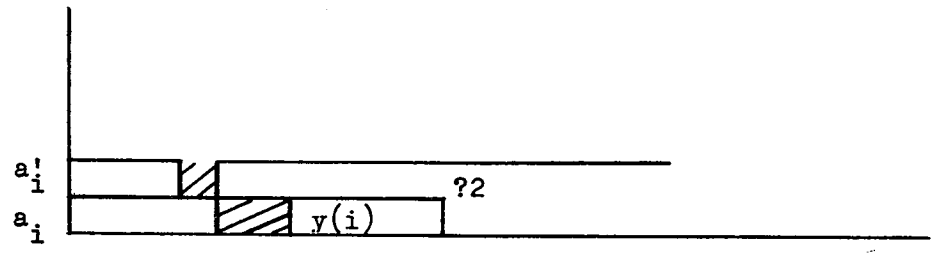
$\sigma'(y(i)) = \sigma(y(i),i,y)$ (cf.première partie du lemme).

Le premier cas est impossible (Lemme 6) le second également (Théorème 00).

DONC, IL N'EXISTE PAS DE TACHES DANS ?1 qui n'est donc qu'une période d'inactivité. Donc, Γ' est de la forme :



On peut donc bâtir un planning Γ", pour lequel les dates d'activa sont identiques à celles de Γ', et pour lequel, l'identité avec Γ est augme d'un cran au moins, en intervertissant l'usage de a_i et a'_i entre lib(a_i,i-1 et l'∞ par rapport à Γ' ce pour tous les couples a_i a'_i.



Par induction, au pas n, on obtient un planning identique à Γ pou lequel les dates d'activation sont celles de Γ'. Donc, Γ' n'est pas meilleu que Γ.

Conclusion.

Tous les plannings générés par la SHP, à l'aide des théorèmes 00 e et de la propriété n° 1 sont actifs.

V. 4. OPTIMALITE DE LA SHP

Définition n° 2.

Si l'on note $\sigma_i(t)$ l'heure de départ de la tâche t, dans un planning Γ_i, alors un planning Γ_i sera dit quasi égal à un planning Γ_2 si $\forall t$, $\sigma_1(t) = \sigma_2(t)$. Par abus on notera la quasi-égalité = .

PROPRIETE N° 3.

La quasi-égalité des plannings est une relation d'équivalence.

Démonstration.

Evidente.

Remarque n° 1.

Le système d'induction utilisé pour le lemme 7 consistait à définir une suite de plannings quasi égaux pour démontrer la quasi-égalité de Γ et Γ'.

Définition n° 3.

Un planning Γ_1 sera dit supérieur ou quasi égal à un planning Γ_2 si $\forall t$, $\sigma_1(t) \leq \sigma_2(t)$; il sera dit supérieur au sens strict si :

$$\forall t, \quad \sigma_1(t) \leqq \sigma_2(t)$$
et
$$\exists t_1, \quad \sigma_1(t_1) < \sigma_2(t_1)$$

Par abus, on note $\geq$ ou $>$ (Noter l'inversion de sens).

PROPRIETE N° 4.

La relation 'supérieur ou quasi égal' est une relation d'ordre partiel

Démonstration.

Evidente.

Définition n° 4.

Un planning Γ sera dit équivalent à un planning Γ' si :

$$\Gamma = \Gamma' \quad \text{ou si} \quad \Gamma > \Gamma'$$

i.e. s'il lui est supérieur ou quasi égal.

PROPRIETE N° 5.

Cette relation n'est pas une relation d'équivalence, l'abus de langage est motivé par le fait que l'on puisse ré-arranger Γ' pour obtenir Γ sans retarder de tâches. Le graphe de la relation n'est pas celui d'une relation d'équivalence, mais le treillis d'un ordre partiel, possédant plusieurs minorants, et pas de borne inférieure. L'idée couverte par cette relation est de ne considérer que les minorants (active-schedules).

PROPRIETE N° 6.

Il n'est pas généré de plannings équivalents.

Démonstration.

Le théorème 2 prouve qu'il n'est pas généré de plannings quasi égaux. Le lemme 7 montre qu'il n'existe pas de planning supérieur à un planning généré. (Tous les plannings générés sont actifs). Il n'en existe donc pas parmi les plannings générés.

THEOREME III.

Tous les minorants (active-schédules, plannings pour lesquels il n'existe pas de planning supérieur) sont générés.

Démonstration.

Soit Γ' un tel planning, soit y la permutation obtenue en classant ce planning $y = \text{Classement}(\Gamma')$, soit Γ le planning obtenu par la SHP à partir de y :

$$\Gamma = \text{SHP}(y).$$

Reprenons les notations utilisées pour le lemme 7.

$y = \{y(1), \ldots, y(n)\}$ la permutation ;

$\{\sigma(y(1),1,y), \ldots, \sigma(y(n),n,y)\}$ les heures de départ selon la SHP ;

$\{mach(y(1)), \ldots, mach(y(n))\}$ les ressources allouées selon la SHP ;

$\{\sigma'(y(1)), \ldots, \sigma'(y(n))\}$ les heures de départ dans Γ' ;

$\{mach'(y(1)), \ldots, mach'(y(n))\}$ les ressources allouées selon Γ'.

La permutation y est par hypothèse compatible avec les relations de précédence, sinon Γ' ne le serait pas.

Soit i le premier entier tel que :

$$\sigma'(y(i)) < \sigma(y(i),i,y)$$

ou

$$\sigma'(y(i)) > \sigma(y(i),i,y)$$

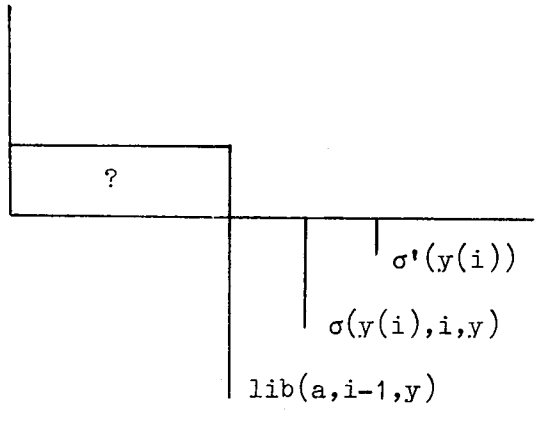
ou

$$mach(y(i)) \neq mach'(y(i))$$

ou tel que la génération soit arrêtée.

1) Si $\sigma(y(i),i,y) < \sigma'(y(i))$ uniquement $(mach(y(i)) = mach'(y(i)))$ alors, il y a une période d'inactivité inutile dans le planning Γ'.

Considérons le diagramme d'occupation d'une machine "a" de $mach(y(i))$ après avoir placé $y(1) \ldots y(i-1)$.



La SHP nous dit qu'elle est libre à un temps :

$$lib(a,i-1,y) \leq \sigma(y(i),i,y) < \sigma'(y(i))$$

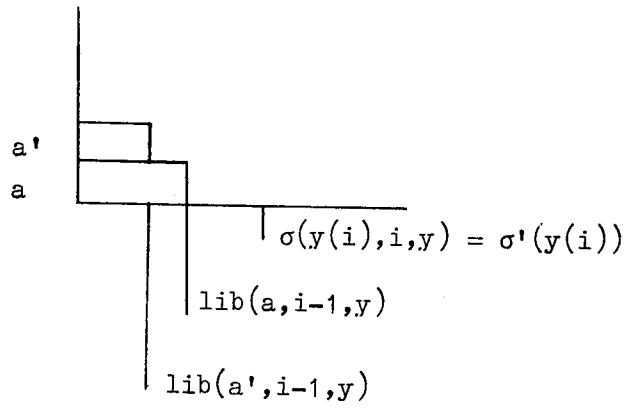
et que de plus, il en est ainsi pour toutes les machines appartenant à mach(y(i)) ; et que tous les prédécesseurs de y(i) sont déjà placés et terminés avant $\sigma(y(i),i,y)$.

Il est donc inutile de faire attendre y(i) jusqu'au temps $\sigma'(y(i))$, d'autant qu'aucune tâche ne sera placée dans ce temps mort par Γ', puisqu'une telle tâche aurait alors un rang inférieur à i dans y et serait donc déjà placée.

2) Si l'on a $mach(y(i)) \neq mach'(y(i))$ uniquement $(\sigma(y(i),i,y) = \sigma'(y(i)))$.

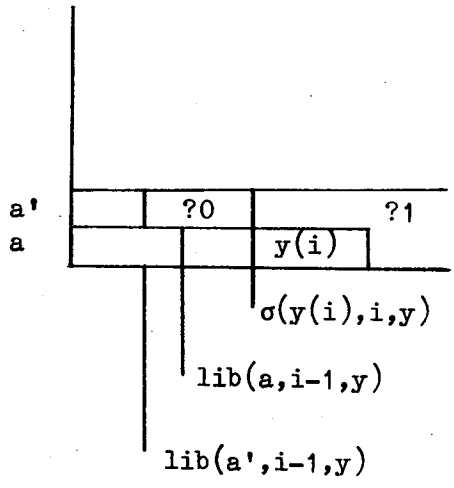
Considérons une paire de machines de même type, a, a'. a étant allouée par la SHP, a' par Γ'.

Examinons le diagramme d'occupation de ces deux machines.

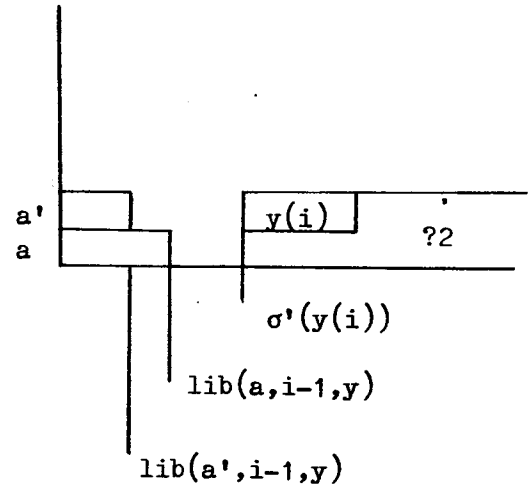


avec $lib(a',i-1,y) \leq lib(a,i-1,y) \leq \sigma(y(i),i,y) = \sigma'(y(i))$

α) Etat après avoir placé y(1) ... y(i-1) selon l'une quelconque des deux politiques SHP ou Γ'.



β) Etat final selon la SHP.



γ) Etat final selon Γ'.

?0, ?1, ?2, sont trois ensembles inconnus de tâches ou de périodes d'inactivité. On note que pour Γ' il ne peut y avoir de tâche nouvelle avant y(i), puisqu'elle aurait alors un rang inférieur à i.

?0 sert à tenir compte d'une possible amélioration due à la SHP et qui entraînerait alors un arrêt de la génération. Nous traiterons ce cas en son temps, ne mentionnant ?0 que pour la rigueur du dessin.

On constate que l'on peut inverser dans Γ' l'utilisation de a et a' entre les temps $t = \sigma'(y(i))$ et $t = \infty$.

En faisant de même pour tous les couples a, a', on obtient un plannin Γ" quasi égal à Γ', et identique à Γ au moins un pas de plus,avec lequel on peut poursuivre la démonstration.

3) Si $\sigma'(y(i)) < \sigma(y(i),i,y)$.

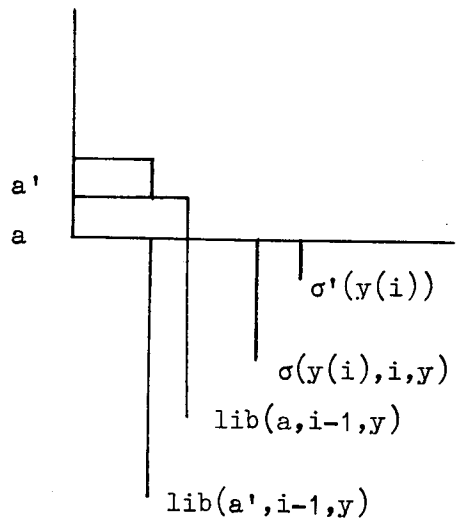
Cela signifie que y(1) ... y(i-1) étant placées il existe une date de départ possible inférieure à celle fournie par la SHP. Il ne peut alors s'agir que d'un trou.

Mais d'une part l'existence d'un trou est impossible (lemme 6), d'autre part si y(i) était placée dans un trou son rang serait inférieur à i, ce qui est également impossible.

4) Si $\sigma'(y(i)) > \sigma(y(i),i,y)$ et $mach(y(i)) \neq mach'(y(i))$.

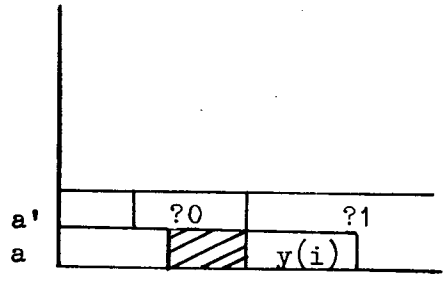
4.1 : Considérons tout d'abord le cas des ressources.
Soit a, a' une des paires ... deux cas possibles :

4.1.1 : a' n'est pas allouée par la SHP, parce que libérée trop tôt.
Ce qui correspond aux diagrammes d'occupation suivants :

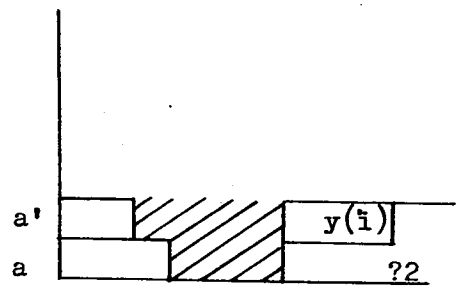


avec : $lib(a',i-1,y) \leq lib(a,i-1,y) \leq \sigma(y(i),i,y) < \sigma'(y(i))$

α) Après avoir placé y(1) ... y(i-1) selon l'une quelconque des deux politiques.



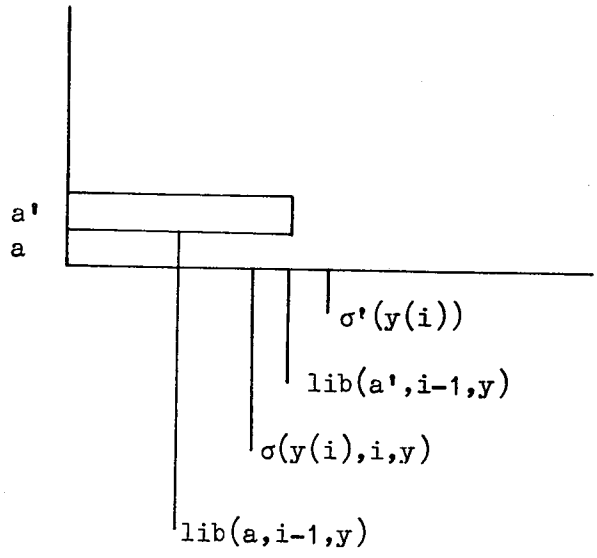
β) Etat final selon la SHP.



γ) Etat final selon Γ'.

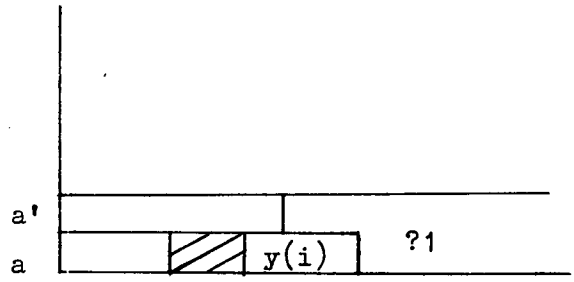
Les mêmes remarques qu'en "2) de ce théorème" sont valables, on peut donc dans Γ' inverser l'usage de a et a'.

4.1.2 : a' n'est pas allouée par la SHP parce que libérée trop tard. Ce qui correspond aux diagrammes d'occupations suivants :

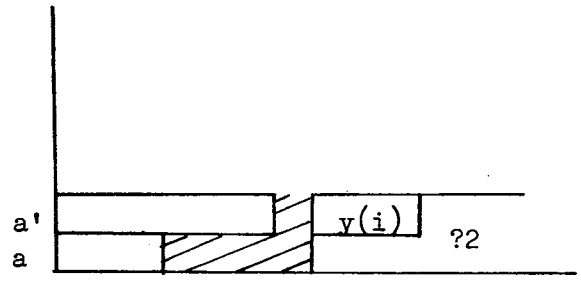


avec : $lib(a,i-1,y) \leq \sigma(y(i),i,y) < lib(a',i-1,y) \leq \sigma'(y(i))$.

γ) Etat après avoir placé y(1), ... y(i-1) selon l'une quelconque des deux politiques.



β) Etat final selon la SHP.



γ) Etat final selon Γ'.

Les mêmes remarques qu'en "2) de ce théorème" sont valables, on peut donc dans Γ' inverser l'usage de a et a'.

Dans les deux cas (4.1.1. et 4.1.2.) on peut donc inverser dans Γ' l'usage de a et a' de façon à obtenir un planning Γ", quasi égal à Γ', ne différant plus de Γ que par les heures d'activation, cas n° 1 de ce théorème.

5) Si la génération est arrêtée.

5.1 : Elle peut l'être à cause du théorème 0 (Dépassement d'une date critique). Si ce dépassement n'a lieu que dans Γ, on se trouve alors dans le cas 3) ($\sigma' < \sigma$) il y a impossibilité. Si ce dépassement a lieu dans Γ et Γ' alors Γ' ne vérifie pas les dates critiques, et est donc inutile.

5.2 : Elle peut l'être à cause du théorème 00. Ce théorème se traduit par au pas i, il existe y(i+k) tq

$$\sigma(y(i+k),i,y) + d(y(i+k)) \leq \sigma(y(i),i,y)$$

c'est-à-dire qu'il y a un trou dans Γ', ou que $\sigma(y(i),i,y) > \sigma'(y(i))$. Dans le premier cas (trou) Γ' n'est pas minorant, dans le second, il y a impossibilité (3)).

5.3 : Elle peut l'être à cause de la propriété n° 1.

C'est-à-dire $\sigma(y(i),i,y) < \sigma(y(i-1),i-1,y)$

ou

$$\begin{cases} \sigma(y(i),i,y) = \sigma(y(i-1),i-1,y) \\ \text{et} \\ y(i) < y(i-1) \end{cases}$$

Les quatre premières parties de ce théorème prouvent que Γ et Γ' sont quasi égaux jusqu'à l'arrêt de la génération donc, pour que la propriété n° 1 soit vérifiée par Γ, alors qu'elle ne l'était pas par Γ' (sinon y serait différente) il faut que $\sigma(y(i),i,y) < \sigma'(y(i))$ ce qui est impossible (1)).

Conclusion.

Les quatre premières parties de cette démonstration ont prouvé que l'on pouvait définir une suite de plannings Γ', Γ'', ... $\Gamma^{(n+1)}$ quasi égaux et identiques à Γ jusqu'au pas 0, 1, ... n ; si la génération de Γ n'était pas arrêtée. La cinquième partie a prouvé que si la génération de Γ était arrêtée, Γ' n'était pas minorant. Ce que nous savions déjà, ayant démontré que tout planning éliminé l'était à juste titre. Il semblait néanmoins utile de refaire une démonstration directe dans ce cas particulier pour la clarté et la rigueur du théorème.

Donc, nous avons démontré que tout minorant était généré, après avoir démontré que seuls des minorants l'étaient, et qu'ils ne l'étaient qu'une seule fois.

PROPRIETE N° 7.

La SHP et son inverse, la fonction classement, établissent une bijection entre l'espace des permutations non éliminées, et celui des plannings minorants.

Démonstration.

* Le théorème 2 a montré que la SHP était injective.
* Le lemme 7 a montré que son image était incluse dans l'espace des minorants.

* Le théorème 3 a montré que la SHP était surjective sur cet espace.
* La propriété n° 2 montre que Classement = SHP^{-1} partout ou SHP est définie (la permutation n'est pas éliminée).

Remarque.

Il est intéressant de noter que même sans la SHP, (essai exhaustif de toutes les allocations possibles) la plupart des résultats restent valables ; en particulier, les critères d'élimination et la propriété de surjection.

Mais dans ce cas, ce ne sont plus deux plannings quasi égaux qui sont identifiés, et l'un deux qui est éliminé, mais un planning inférieur à x autres qui est repéré et éliminé, l'image n'étant plus incluse dans l'ensemble des minorants.

De plus, le critère d'élimination perd de sa puissance, tous les plannings non actifs ne sont pas éliminés, et si l'injectivité subsiste, c'est parce que les deux images de deux permutations menant à un planning et à un minorant équivalent ne sont pas confondues. Si l'injectivité subsiste donc par rapport à la quasi-égalité, elle n'existe plus par rapport à la relation 'supérieur à':

i.e. : $\nexists\ y,y' \qquad y \neq y' \qquad GEN(y) = GEN(y')$

$\exists\ y,y' \qquad y \neq y' \qquad GEN(y) < GEN(y')$

ce qui n'est pas le cas avec la SHP (propriété n° 6).

Conclusion.

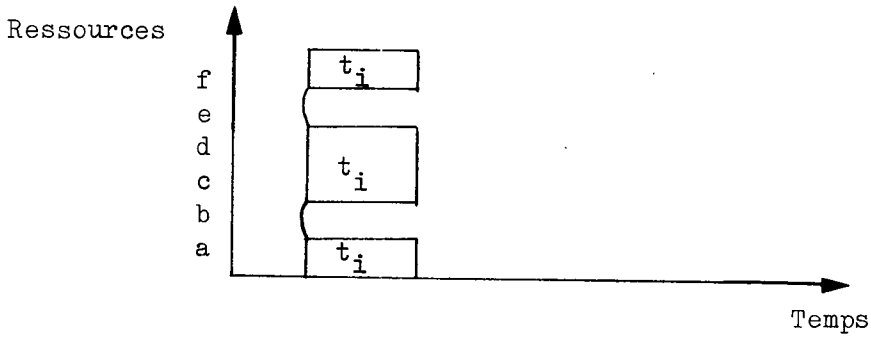
Le lemme 1 était une première justification de la SHP, il impliquait l'acceptation d'un axiome, celui de la surjection des méthodes de génération par étude de permutations.

Nous avons maintenant démontré cette surjection. Nous avons de plus démontré l'inclusion de l'image dans l'espace des plannings minorants et la bijection établie par la SHP.

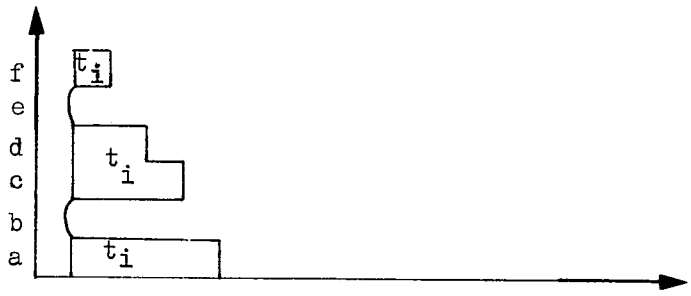
V. 5. EXTENSION DU PROBLEME : LES DEMANDES TARDIVES

V. 5. 1. Les relâches décalées.

Jusqu'à présent, nous avons considéré que toute tâche demandait toutes ses ressources au début de son activation, et les conservait pour toute la durée de cette activation. Cette idée correspond à des diagrammes d'occupation par les tâches de la forme :



Une première extension possible est celle où toutes les ressources sont demandées au début de l'activation, et relâchées au cours de cette activation, ce qui correspond au diagramme d'occupation suivant :



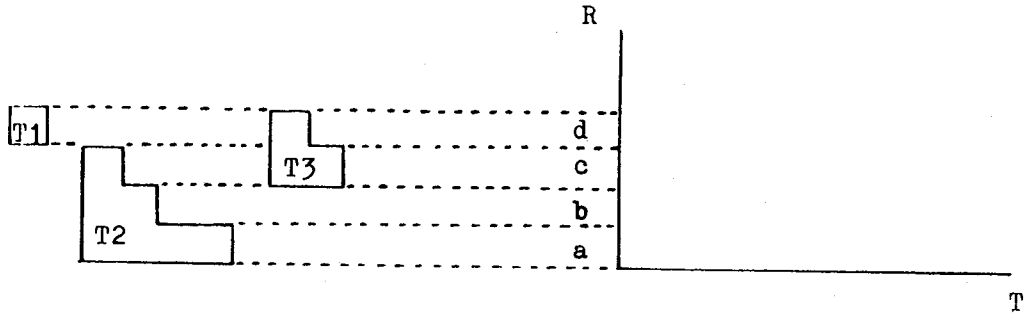
Cette extension simple nous fait pourtant perdre le bénéfice de l'utilisation du théorème 00, faute de savoir quelle durée d(t) choisir (cf. Ex 1 et 2). Il n'existe pas de méthode simple de récupération des trous menant à un optimum, toutes obligent à un réordonnancement partiel (Ex 3). Cependant, la récupération d'un trou entraînant l'application de la propriété n° 1, une seule récupération par permutation est au plus nécessaire.

Illustration.

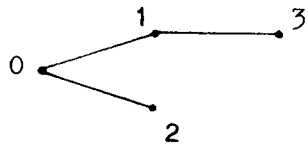
Exemple 1.

Les trous peuvent être préjudiciables.

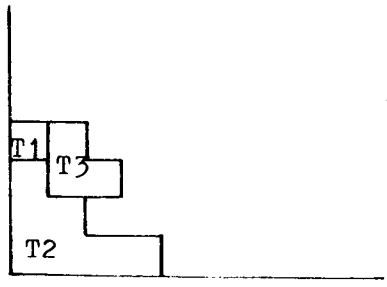
Demandes : les ressources sont toutes différentes.



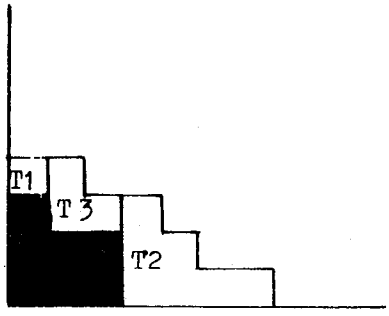
Précédence :



Pas 1. Permutation 1-2-3 :



Pas 2. Permutation 1-3-2 :

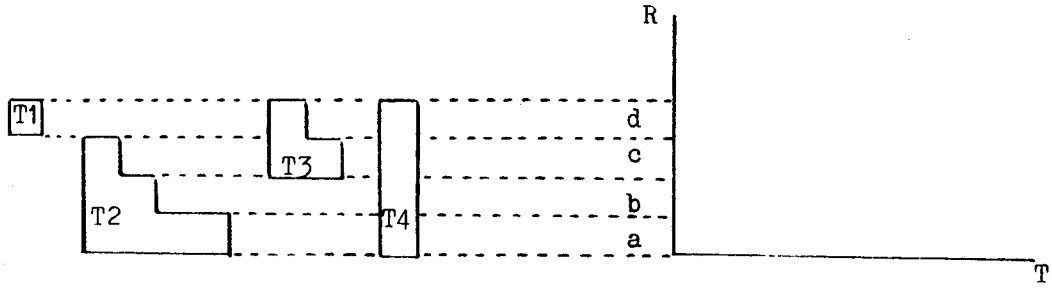


Si l'on prend pour le théorème 00 la durée maximale, faute d'avoir su remarquer que T2 ne retardait pas T3, on crée un trou préjudiciable à T2.

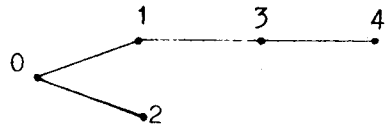
Exemple 2.

Les trous peuvent être bénéfiques.

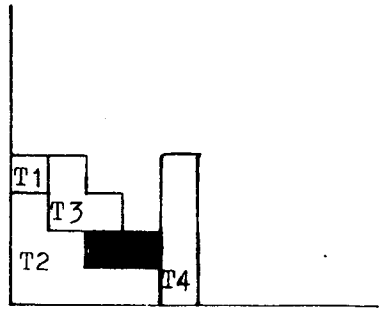
Demandes : les ressources sont toutes distinctes.



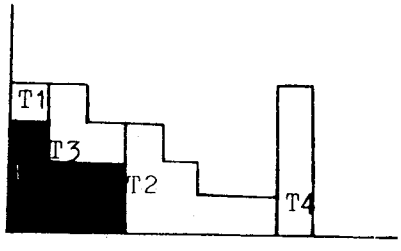
Précédence :



Pas 1. Permutation 1-2-3-4 :

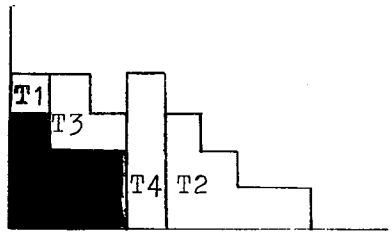


Pas 2. Permutation 1-3-2-4 :



On crée un trou préjudiciable à T2 et T4.

Pas 3. Permutation 1-3-4-2 :

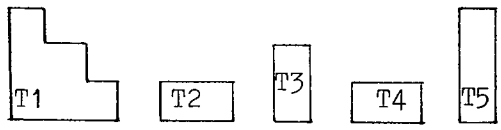


Le même trou est cette fois bénéfique à t4.

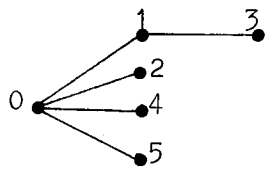
Exemple 3.

Récupération des trous.

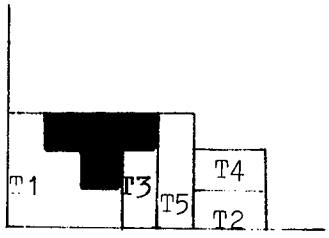
Demandes : les ressources sont toutes équivalentes. On dispose de trois unités.



Précédence :

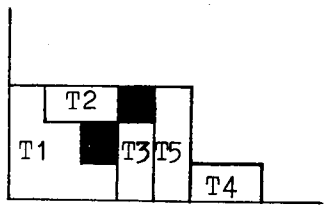


Pas 1. Permutation 1-3-5-2-4 sans récupération :



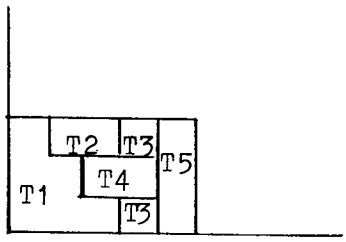
Il y a un trou préjudiciable à T2 et T4.

Pas 2. Permutation 1-3-5-2-4 avec récupération :



La récupération n'est pas parfaite, puisqu'il reste un ensemble de deux trous préjudiciables à T4. L'optimum n'est pas atteint, mais cette récupération suffit à arrêter la génération conformément à la propriété n° 1.

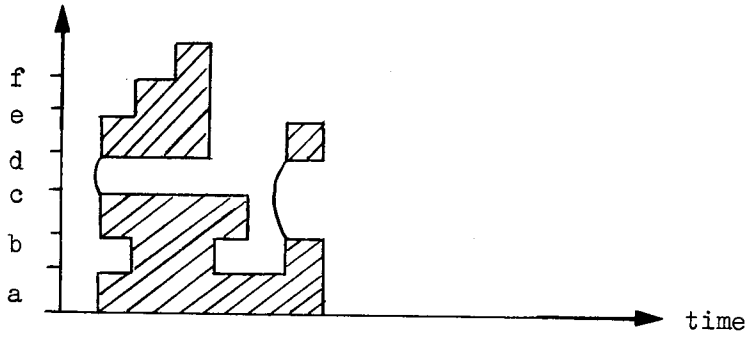
Pas 3. Permutation 1-2-4-3-5 :



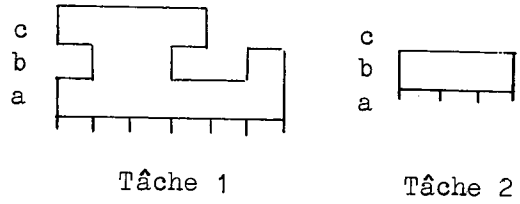
Un optimum.

V. 5. 2. Les demandes décalées.

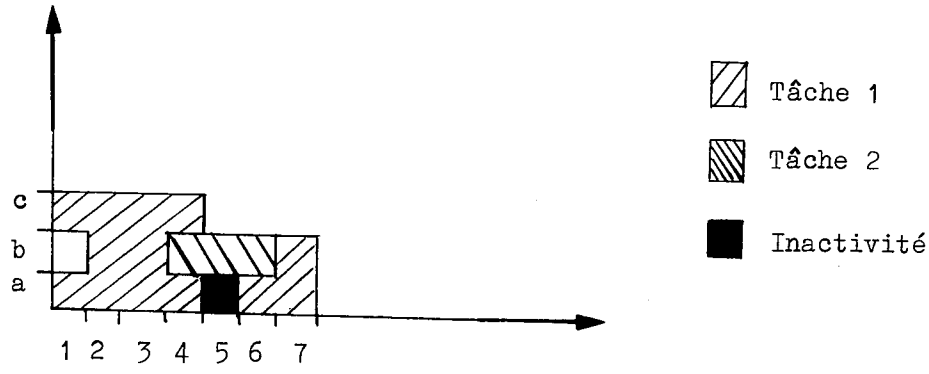
Les choses deviennent plus compliquées quand les demandes prennent la forme la plus quelconque :



On constate alors que si l'on considère la tâche comme un bloc monolithique, on ne peut obtenir les plannings optimaux :



Les tâches 1 et 2 ne peuvent être planifiées en moins de huit unités de temps si on les laisse sous cette forme. Mais si on arrête l'exécution de 1, sans lui retirer ses ressources, on peut les exécuter en sept unités de temps :



On notera qu'il ne s'agit nullement de préemption, la ressource "a'" n'est pas retirée à la tâche 1. L'activité est simplement arrêtée.

On remarquera également que l'utilisation de tels temps morts implique qu'ils ne gênent en rien l'exécution. C'est le cas dans un calculateur où la mémoire peut rester occupée mais inactive, où un fichier, un périphérique peuvent rester alloués mais inactifs, où un processeur peut rester inactif, ou faire de l'attente active. Il faut être plus circonspect pour une production industrielle ; si un conteneur peut stationner durant des jours, la cuisson d'une pièce est une opération qu'il n'est pas possible d'arrêter.

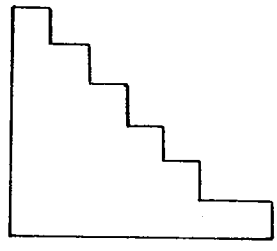
On constatera enfin, qu'utiliser de tels temps morts implique que l'on sait les placer au mieux. Dans l'exemple précédent, on aurait obtenu le même résultat en plaçant un temps mort aux temps 4 ou 6, plutôt qu'au temps 5. Quelques réflexions vont nous aider à mieux les utiliser.

LEMME N° 8.

Si les heures de demande sont identiques, l'usage des temps morts est inutile et préjudiciable.

Démonstration.

Un temps mort ne servirait qu'à augmenter la durée d'occupation des machines, sans pour cela permettre une allocation plus précoce de l'une d'entre elles.



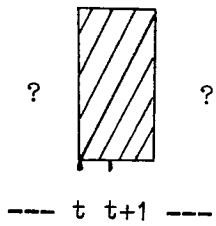
Dans un tel cas, les temps morts sont inutiles.

LEMME N° 9.

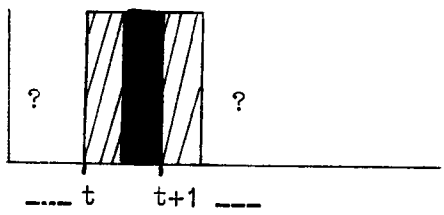
si l'on a le choix entre deux périodes t et t+1 contigües, où les mêmes ressources sont occupées, on peut choisir de placer le temps mort avant t+1.

Démonstration.

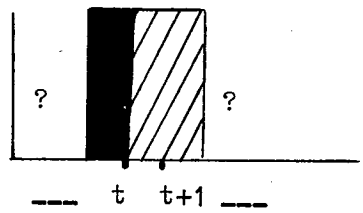
Considérons les diagrammes d'occupation :



α) La tâche considérée. On ne considère que les t et t+1ème instants d'activité.



β) Le temps mort est placé avant l'instant d'activité t+1.



γ) Le temps mort est placé avant l'instant d'activité t.

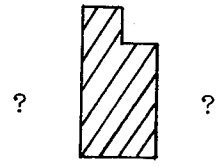
La démonstration est évidente, le résultat global est le même, ce distinguo utile à l'ordonnancement disparaît à l'exécution.

LEMME N° 10.

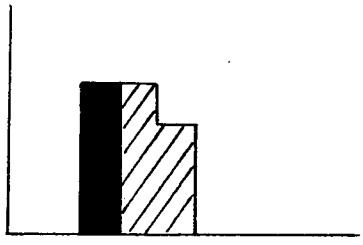
Si l'on a le choix entre deux périodes, t et t+1, contigües, et où les ressources occupées en t+1 sont un sous-ensemble de celles occupées en t, alors on a intérêt à choisir t+1.

Démonstration.

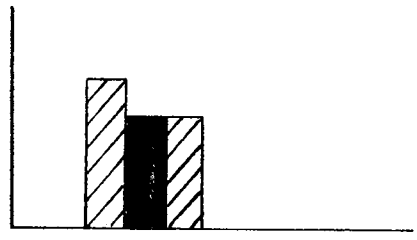
Considérons les diagrammes d'occupation :



α) La tâche considérée.



β) Le temps mort est placé avant l'instant d'activité t.



γ) Le temps mort est placé avant l'instant d'activité t+1.

La démonstration est évidente.

LEMME N° 11.

On peut intervertir temps mort et activité si les ressources occupées dans les deux cas sont les mêmes.

Démonstration.

Evidente.

THEOREME 4.

On peut ne placer les temps morts qu'avant une demande de ressources.

Démonstration.

Considérons l'instant d'activité t avant lequel on veut placer le temps mort. Considérons l'instant t+i de la prochaine demande.

Si i=0 C.Q.F.D.

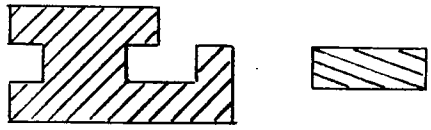
Sinon : Si en t+1 les mêmes ressources sont utilisées, le lemme 9 joue. On place le temps mort avant t+1 et l'on recommence.

Si en t+1 c'est un sous-ensemble des ressources qui est utilisé. Alors le lemme 10 joue, et l'on recommence.

Si en t+1 il y a une demande, alors le lemme 11 joue et C.Q.F.D.

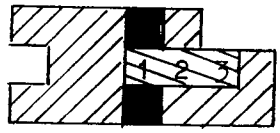
Illustration.

Sur l'exemple des tâches 1 et 2 déjà utilisé.



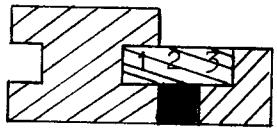
1 2 3 4 5 6 1 2 3

On peut placer le temps mort avant 4.



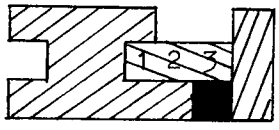
1 2 3 4 5 6

Mais en 5 les ressources sont un sous-ensemble, donc lemme 10 :



1 2 3 4 5 6

Mais en 6 il y a une demande, donc lemme 11.



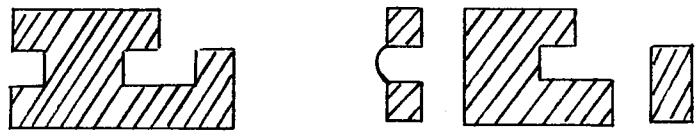
1 2 3 4 5 6

Conclusion.

Ce théorème sert à limiter le nombre de points où seront éventuellement placés les temps morts, et nous permet de trouver un juste milieu entre le bloc monolithique non optimal et la considération unité de temps par unité de temps quasi incalculable.

Nous considérerons maintenant chaque tâche comme une suite de sous-tâches qui peuvent éventuellement être séparées par des temps morts. Chaque sous-tâche débutant par une demande de ressources et finissant avant une demande de ressources.

Exemple : la tâche 1 devient avec ce système.



V. 6. LES PROBLEMES POSES PAR L'EXTENSION DE L'ALGORITHME ET LEUR SOLUTION

V. 6. 1. Exposé des problèmes de base.

Le problème de la non-allocation des ressources libérées par une sous-tâche, mais pas par la tâche, va nous amener à reconsidérer l'algorithme.

Dans la première partie, il n'y avait aucun risque d'étreinte fatale puisque chaque tâche demandait toutes ses ressources en même temps.

L'ordonnancement sans partition en sous-tâches s'il ne pose pas le problème de l'étreinte fatale, pose celui de la récupération des trous, et de l'optimalité sans temps morts, et par-là même remet en question la SHP.

L'ordonnancement avec partition en sous-tâches pose le problème de la non-allocation des ressources en attente active (temps-morts) et remet en question le critère d'élimination du théorème 00. En effet, la durée des tâche devient variable, puisque le nombre de temps morts et leur durée sont inconnus

V. 6. 2. La solution des problèmes de base.

En posant que la date de libération d'une ressource allouée à une sous-tâche, et réservée pour une autre sous-tâche, donc éventuellement en temps-mort, est considérée comme infinie par toutes les tâches ou sous-tâches différentes de celle pour laquelle elle est réservée, on dispose d'une solution au problème de la non-allocation sans avoir à trop modifier l'algorithme.

Une solution pour l'élimination consiste à ne faire jouer le théorème 00 que pour la dernière sous-tâche d'une chaîne. On perd alors l'injectivité par rapport à la relation, supérieure à et l'inclusion de l'image parmi les minorants, mais on conserve la surjectivité, qui ne serait pas garantie par l'application sans discernement du théorème 00.

V. 6. 3. Un problème du second degré : les étreintes fatales.

V. 6. 3. 1. Exposé du problème.

Un nouveau prolème apparaît, celui des étreintes fatales, puisqu'une chaîne peut bloquer une ressource avant d'en demander une autre. Il est donc possible au cours de l'algorithme d'arriver à une configuration où toutes les tâches de av(i) s'attendent les unes les autres par l'intermédiaire de ressource bloquées.

V. 6. 3. 2. Considérations générales.

Nous pouvons pour résoudre ce problème tirer parti du fait que l'ordonnancement est statique pour éviter les deux solutions non optimales usuelles (demandes groupées ou algorithme du banquier) et choisir parmi deux autres :

- la détection multiple,
- la prévention par mémorisation,

nous ne cherchons pas à prévenir la première apparition d'un cas d'étreinte fatale. Si les cas sont rares, on peut se contenter de les détecter à chaque fois, même si le même cas se retrouve dans divers plannings. Si les cas sont nombreux, on mémorise les cas d'étreinte fatale pour prévenir la réapparition du même cas dans divers plannings.

V. 6. 3. 3. La détection des étreintes fatales.

Une matrice binaire, quelle que soit sa représentation interne (listes, listes-croisées, tableaux...), et la méthode de MARIMONT [2] semblent être le système de détection le plus simple et le moins coûteux.

V. 6. 3. 4. La récupération des étreintes fatales.

Lorsqu'une étreinte fatale est détectée, il est nécessaire pour pouvoir poursuivre l'algorithme de libérer une des ressources engagées dans le cercle de l'étreinte ; pour ce faire, il faut remonter dans l'arbre au moins jusqu'au point où la dernière des ressources en cause a été allouée. Tout successeur de ce nœud menant au même problème. De plus, tant qu'une au moins des tâches mises en cause n'a pas été ordonnancée, il est inutile de réordonnancer la chaîne désordonnancée. (cf.Lemme 12).

LEMME N° 12.

Pour résoudre un problème d'étreinte fatale, il faut désordonnancer une des chaînes mises en cause, et ne pas la réordonnancer avant l'ordonnancement d'une des sous-tâches mises en cause.

Démonstration.

Représentons les chaînes de sous-tâches mises en cause de la façcn suivante :

Chaîne 1 $A_{11}, A_{12} \quad \cdots \quad , A_{1j_1}, B_1, C_{11}, \quad \cdots \quad C_{1k_1}$

$\vdots$

Chaîne 2 $A_{i1}, A_{i2} \quad \cdots \quad , A_{ij_i}, B_i, C_{i1}, \quad \cdots \quad C_{ik_i}$

$\vdots$

Chaîne n $A_{n1}, A_{n2} \quad \cdots \quad , A_{nj_n}, B_n, C_{n1}, \quad \cdots \quad C_{nk_n}$

Où les B_i sont les sous-tâches mises en cause, avec B_i veut la machine M_{i+1} allouée à B_{i+1} et $n+1 = 1$; les A_{ix} les sous-tâches utilisant ces ressources avant les B_i, les A_{i1} étant celles qui les ont demandées. La machine M_i étant donc allouée à la chaîne i de A_{i1} à B_i au moins.

Si à un pas de l'algorithme, les sous-tâches :

$A_{1x_1}, A_{2x_2} \quad \cdots \quad , A_{ix_i}, \quad \cdots \quad A_{nx_n}$ sont ordonnancées,

que se passe-t-il aux pas suivants.

On peut ordonnancer des tâches x,y non liées aux chaînes considérées.
On peut ordonnancer des tâches A_{xy}.
On ne peut ordonnancer les B_i puisqu'ils veulent une machine déjà allouée.

Mais comme le nombre de tâches est fini, il va arriver un pas où il n'y aura dans av(i) que les tâches B_i, $i:1\rightarrow n$; à moins qu'une autre situation d'étreinte fatale n'apparaisse avant.

Donc pour ne pas retomber dans la même situation, il faut revenir en arrière jusqu'au pas où la dernière des A_{x1} a été ordonnancée. De plus on peut déjà dire que l'on ne peut la réordonnancer tant qu'un C_{y1} ne l'aura pas été. Mais il s'agit là déjà de prévenir la réapparition d'un cas identique dans un autre sous-arbre.

V. 6. 3. 5. La prévention des étreintes fatales.

La dernière partie de VI. 3. 4. était déjà une introduction à ce problème. Si l'on sait que l'ordonnancement simultané de certaines sous_chaînes a mené à une étreinte fatale, on doit éviter de recommencer la même erreur pour un autre sous-arbre.

Une méthode simple de prévention de la répétition des cas d'étreinte fatale peut être :

A chaque fois qu'une tête de chaîne Az1 est ordonnancée, on ajoute son numéro de chaîne z à la liste des chaînes en cours.

A chaque fois qu'une queue de chaîne Czk est ordonnancée, on enlève son numéro de chaîne de la liste des chaînes en cours.

A chaque fois qu'une étreinte fatale a lieu, on mémorise la liste des chaînes mises en cause.

A chaque fois qu'une tête de chaîne est ordonnancée, après avoir ajouté son numéro on regarde si la liste des chaînes en cours contient une sous-liste égale à une liste d'étreinte fatale.

Il faut noter que de la façon dont nous avons fait le partitionnement en sous-tâches, toutes les sous_tâches sont des têtes de chaînes au sens des étreintes fatales ; et que, lors de la détection d'une telle étreinte, il faut mémoriser les têtes de chaîne correspondant à l'étreinte et non le début de la tâche.

V. 6. 4. Retour à la méthode générale. Comparaison des coûts.

Nous avons vu qu'en deux cas au moins, le théorème 00 n'était pas applicable (cf.§ V.5.1. et V.6.1.). Aussi, pour conserver l'injectivité par rapport la relation "supérieur à", nous sommes obligés de faire une recherche de trous avec les conséquences sur le coût des solutions que cela implique.(cf.Chap.IV § IV.5.8.).Il faut donc se montrer circonspect quant à l'utilisation d'une telle méthode ; néanmoins, c'est la seule qui ,dans les cas où le théorème 00 n'est pas utilisable ,permette un plein rendement de la propriété n° 1, et donc garantisse l'injectivité. Il faut donc mesurer les pertes dues à ce travail supplémentaire, et à l'augmentation des besoins en mémoire centrale, et les comparer avec le coût des plannings générés inutilement. En particulier, quand le nombre de tâches n croît, les besoins en mémoire centrale pour les zones de travail de la SHP sans récupération restent constants et égaux au nombre de ressources ceux du système avec récupération sont 2n fois plus grands. Pour ce qui est du temps, la récupération multiplie le temps nécessaire à trouver les plannings actifs par la durée du questionnaire ; la non-récupération par le pourcentage de régénération.

Récapitulatif des coûts comparés.

Données :

n : nombre de tâches ou de sous-tâches,

j : nombre de types de ressources,

r : nombre total de ressources.

G : Graphe de précédence.

HA : Heures d'arrivée vecteur 1 × n.

HDL : Dates critiques - vecteur 1 × n.

W : Demandes — Matrices J × n

D : Durée — Vecteur 1 × n

} ou n listes

Cas de base. Cas des extensions.

Zone de travail pour les résultats intermédiaires :

AV : n listes d'au plus n éléments dont une seule doit être résidente.

Planning : Pour la SHP sans récupération, n vecteurs à r éléments (dates de libération) dont 1 seul doit être résident.

Pour la récupération 2 × n vecteurs à r éléments tous résidents.

Durée :

- Temps de base pour fournir les plannings actifs :

$f(G,HDL,W,D)$

$kn \leq f \leq kn!$ k : constante égale à la durée des opérations à chaque pas.

- Temps de calcul :

Sans récupération :

f si le théorème 00 est applicable ;

f(1+C) si le théroème 00 n'est pas applicable

C : coefficient de régénération.

Avec récupération :

f*dq dq : durée du questionnaire pour l'examen du planning.

V. 6. 5. Deux extensions possibles, non résolues.

Deux problèmes complémentaires, mais hélas non résolus, sont dignes d'intérêt.

α) La non-stabilité des types de ressources, qu'il s'agisse de la variation du nombre de ressources, de la préférence d'une tâche pour une des ressources, ou de l'équivalence de deux ressources par rapport à une tâche uniquement.

β) Les temps de transition non nuls, ou problème équivalent, la durée fonction de la ressource allouée.

Dans les deux cas, seul l'essai systématique de toutes les combinaisons possibles de ressources, ou bien un réordonnancement continuel garantissent la surjectivité. Néanmoins, il semble que ces problèmes ne soient actuellement solvables qu'avec l'aide d'un opérateur humain [3], et dans ce cas, la SHP fournit une excellente solution de départ, en particulier, grâce au fait qu'elle indique nominativement quelles ressources sont allouées, et ce à chaque pas, ce qui permet un traitement conversationnel.

CONCLUSIONS SUR LA SHP

Nous avons introduit une méthode de choix des ressources, la SHP, qui nous permet, à chaque pas, de ne considérer que la frontière du planning généré, et non son ensemble, et de spécifier le nom des ressources allouées.

Au § V.6.4. nous avons donné un tableau récapulatif du gain en temps et en place mémoire centrale dû pour le temps à la non-considération de l'ensemble du planning, pour la place, au fait que le programme devient une procédure récursive dont les résultats intermédiaires peuvent être stockés dans une mémoire secondaire de type bande magnétique et donc, que la place requise en MC est indépendante du nombre de tâches (au stockage des données du problème près).

α) Dans les cas où le théorème 00 est applicable, en particulier pour le problème de base, ce gain est un gain vrai, la SHP étant injective par rapport à l'équivalence des plannings.

La SHP est donc justifiée comme méthode optimale d'allocation des ressources pour la planification statique par étude de permutations.

De plus, grâce au théorème 0, seuls les plannings vérifiant les dates critiques sont générés, de plus, à tout planning généré, on peut associer un ensemble de dates critiques qu'il est le seul à vérifier, tous les plannings générés sont donc utiles dans le cadre du temps réel. (Ceci ne serait pas le cas pour une recherche de temps total minimal par exemple). Il n'est hélas pas possible pour l'instant d'orienter le parcours de l'arbre de façon à arriver plus vite à une solution, (éliminer le moins de solutions possible avant d'en trouver une).

On peut néanmoins dire que dans l'état actuel des connaissances, la SHP est la meilleure méthode de recherche de solutions ; puisque seules les méthodes combinatoires peuvent les trouver à coup sûr, et que la SHP est parmi celles-ci celle qui ne fournit que des solutions optimales, qu'une fois et une seule, et au meilleur prix.

Il reste donc à fournir des résultats déterministes ou statistiques sur l'odre optimal d'examen des permutations pour obtenir soit un algorithme de recherche directe, soit une méthode donnant statistiquement un temps de sortie de la première solution le plus court.

β) Dans le cas des deux premières extensions, le théorème 00 n'est pas applicable, l'injectivité de la SHP par rapport à l'équivalence des plannings n'étant plus vérifiée, il faut tenir compte des pertes en temps dues aux générations multiples, et les comparer aux gains en temps et place mémoire.

γ) Dans le cas des deux extensions non résolues, la SHP est une base solide pour le travail de l'opérateur humain, en particulier lors du déroulement conversationnel [3].

REFERENCES

[1] SHRAGE L. "Solving resource constrained network problems by implicit enumeration. Non preemptive case". Opns. Research. Vol.18, pp.263-278. (1970).

[2] MARIMONT R.B. "A new method of checking the consistency of precedence matrices". Journal ACM. Vol.6, N° 2, pp.164-171. (1959).

[3] RAND CORPORATION.

DREZNER S.M. VAN HRON R.L. "Design considerations for CAMCOS. A computer assisted maintenance planning and control system". Rand Memorandum RM 5255 PR. (July 1967).

GEOFFRION A.M. "Integer programming by implicit enumeration and Balas method". Rand Memorandum RM 4783 PR. (February 1966).

GEOFFRION A.M. "Implicit enumeration using and imbedded linear program". Rand Memorandum RM 5406 PR. (September 1967).

GEOFFRION A.M. "An improved implicit enumeration approach for integer programming". Rand Memorandum RM 5644 PR. (June 1968).

PRITSKER A.A.B. WATTERS L.J. "A 0/1 programming approach to scheduling with limited ressources". Rand Memorandum RM 5561 PR. (January 1968).

ANNEXE 1

ORDON

Un algorithme pour l'ordonnancement de tâches temps-réel sur des ressources non-préemptives.

NOTICE DE PROGRAMME

Version ALGOL-60

RESUME

Cette brochure a pour but de faciliter la mise en œuvre du programm ALGOL-60 ORDON.

Les références scientifiques sont indiquées ;

les limitations, contraintes de présentation, ainsi que les formats de sortie y sont décrits ;

les listings commentés y sont joints.

SOMMAIRE

A1-0) DEFINITION DU PROGRAMME.

Nom de l'auteur du programme : JORRY Alain.
Nom du programme..... : ORDON, un algorithme pour l'ordonnancement de tâches temps réel sur des ressources non préemptives.
Version..... : ALGOL-60.

A1-1) BUTS ET APPLICATIONS.

Calcul de plannings par une méthode d'énumération,réduite par une politique d'allocation directe dite Shortest Hole Policy qui évite les impasses dues à l'allocation et ramène la taille du problème de P_τ^* à $|\tau|!$, (avec τ ensemble des tâches) et par les théorèmes de L. Shrage [1] qui par la technique classique de l'énumération réduisent la complexité de $|\tau|!$ à $x<<|\tau|!$.

A1-2) DEFINITION MATHEMATIQUE DU TRAITEMENT.

Voir Rapport-Laboria n° 64 [2].

---

*

$$P_\tau = \sum_{i=1}^{|\tau|} P_{\tau,i} \qquad \text{si } |\tau| = 0,\ P_\tau = 1$$

$$P_{\tau,i} = \left\{ \prod_{j=1}^{|N/E_i|} \begin{bmatrix} \omega(i,j) \\ \\ |C(i,j)| \end{bmatrix} \right\} \text{X } P_{\tau-\{i\}}$$

(Voir [2]).

avec $P_\tau = |\tau|!$ si $|C(i,j)| = \omega(i,j)$

La SHP permet de supprimer le premier terme du produit en indiquant la combinaison à choisir parmi toutes celles qui sont possibles et de réduire le problème.

A1-3) LIMITATIONS NUMERIQUES.

Nombre de tâches ≤ limite fixée pour les entiers $(|\tau|)$

Nombre de types de machines ≤ limite fixée pour les entiers $(|N/E_i|)$

Nombre de machines par type ≤ limite fixée pour les entiers $(|C(i,j)|)$

Durée totale des travaux ≤ limite fixée pour les entiers $(\sigma(y(n),n)$

Durée d'une tâche ≤ limite fixée pour les entiers $(\sup(d(i)))$

Nota : La procédure IMPRIM de sortie des résultats est prévue pour une limite de 999, le cas échéant modifier le format de sortie.

A1-4) LISTE DES DIFFERENTS JEUX DE CARTES.

Outre la version présente dite tout en mémoire qui a une occupation de $5|\mathcal{C}|^2$, une version à pile externe d'une occupation de $5|\mathcal{C}|$ est en cours de rédaction [7].

Les deux versions étant rédigées en Algol-60 sont disponibles sur toute machine disposant de ce langage.

A1-5) ENTREES/SORTIES.

Les entrées se font sur le pseudo-périphérique 1, les sorties sur le pseudo-périphérique 6. Le langage de contrôle permet d'associer n'importe quel périphérique réel ou fichier à ces numéros.

A1-6) ENCOMBREMENT.

Dans cette version actuelle à pile externe, la compilation, l'éditio de liens, l'exécution requièrent moins de 29 K mots, pour les données suivante

50 tâches.

10 types de machines.

5 machines par type.

Voir à ce sujet au paragraphe 11 la liste d'encombrement statique donnée par le compilateur.

A'-7) DESCRIPTION DES DONNEES.

Les données sont lues sur cartes (*) par l'instruction READ, le forma est donc celui propre à cette instruction.

Les nombres écrits en décimal, avec ou sans les zéros non significatifs, sur au plus 11 chiffres, sont séparés par un blanc 'ϕ', une virgule ',', un point-virgule ';' ou un commentaire (suite de caractères non numériques) ; et sont écrits dans l'ordre où ils doivent être lus, entre les colonnes 1 à 80 bornes comprises, à raison d'autant de nombres que l'on veut par cartes (y compris 0 si on le désire) avec autant de blancs non significatifs qu'on le désire.

Ex : 1 2 3 peut s'écrire :

col : 1 80

ϕϕ 1ϕ 2ϕ 3ϕ ..... ϕ

ou

ϕ..ϕ1ϕ ... ϕ2ϕ .... ϕ3ϕ .... ϕ

ou

ϕ.ϕ1,2,3 ϕ ..... ϕ

ou

ϕ ... ϕ1ϕ .... ϕ, ϕ ϕ2ϕ
ϕ ... ϕ3

ou

ϕ.....ϕ1.....ϕ
ϕ.....ϕ
ϕ2.....ϕ
ϕ.....ϕ
ϕ.....ϕ
ϕ.....ϕ
ϕ3.....ϕ

N.B. Le graphe de précédence ne doit avoir qu'une seule racine, et celle-ci doit porter le nom de 0. Il doit s'agir d'une tâche fictive car elle ne sert qu'à initialiser l'algorithme, sans être effectivement prise en compte.

(*) Ou dans un fichier, ou à la console, remplacer alors 'carte' par 'enregistrement' ou 'ligne' et 80 par la taille correspondante.

Les données ont la forme suivante (*) :

N	nombre de tâches, la tâche 0 non comprise.
C	nombre de types de ressources.
M	nombre total de ressources.
MAX SUCC	nombre maximum de successeurs pour une tâche.

$M_1 \cdots M_i \cdots M_c$ nombre de ressources de type i $(i : 1 \rightarrow C)$
$\Sigma M_i = M$

$AT_0 \cdots AT_i \cdots AT_N$ heure d'arrivée de la tâche i $(i : 0 \rightarrow N)$
$AT_0 = 0$

$D_0 \cdots D_i \cdots D_N$ durée de la tâche i $(i : 0 \rightarrow N)$
$D_0 = 0$

$Dl_0 \cdots Dl_i \cdots Dl_N$ date critique de la tâche i $(i : 0 \rightarrow N)$
$Dl_0 = 0$

$\omega(0,1) \cdots \omega(0,j) \cdots \omega(0,c)$ Besoins de la tâche 0 en ressources de type j $(j : 1 \rightarrow C)$
$\omega(0,j) = 0$

$\omega(i,1) \cdots \omega(i,j) \cdots \omega(i,C)$ Besoins de la tâche i en ressources de type j $(j : 1 \rightarrow C)$

$\omega(n,1) \cdots \omega(n,j) \cdots \omega(n,C)$ Besoins de la tâche n en ressources de type j $(j : 1 \rightarrow C)$

$P_0 \cdots P_i \cdots P_N$ Nombre de prédécesseurs de la tâche i $(i : 0 \rightarrow N)$
$P_0 = 0$

$S_{01} \cdots S_{0j} \cdots -1$ Liste des successeurs de la tâche 0 (tâches sans prédécesseur réel) suivie du symbole 'fin de liste' : -1

$S_{i1} \cdots S_{ik} \cdots -1$ Liste des successeurs de la tâche i

$S_{N1} \cdots S_{Nl} \cdots -1$ Liste des successeurs de la tâche N

(*) On peut également se référer à l'exemple d'exécution (§ 12).

A1-8) RESULTATS.

Ce programme les donne sous la forme

SCHEDULE

```
AT STEP   1 TASK   1 HAS BEEN SCHEDULED WITH START-TIME    4 .
AT STEP   2 TASK   2 HAS BEEN SCHEDULED WITH START-TIME    9 .
AT STEP   3 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   16 .
AT STEP   4 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   24 .
AT STEP   5 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   30 .
AT STEP   6 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   32 .
AT STEP   7 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   39 .
AT STEP   8 TASK   8 HAS BEEN SCHEDULED WITH START-TIME   39 .
AT STEP   9 TASK  10 HAS BEEN SCHEDULED WITH START-TIME   46 .
AT STEP  10 TASK   9 HAS BEEN SCHEDULED WITH START-TIME   51 .
```

Néanmoins, il est prévu de le coupler avec un second programme pour imprimer les résultats sous forme de diagrammes de Gantt.

A1-9) COLLABORATION AVEC D'AUTRES PROGRAMMES.

Quatre programmes sont rédigés ou en cours de rédaction, qui doivent compléter celui-ci.

- Un programme de vérification de la cohérence des données [5].
- Un programme conversationnel simplifiant l'écriture de ces données, et permettant leur entrée et leur test sous forme de dialogue.
- Un programme d'édition des résultats (cf.§ 8).
- Un programme de génération aléatoire de données [6].

Ne pas oublier que de plus ce programme se présente sous deux formes différentes (tout en mémoire ou pile externe) [7].

A1-10) CHANGEMENTS POUR MODIFICATION DE TAILLE.

Toute variation de 1 à 999 est prise en compte automatiquement grâce aux paramètres de taille N, C, M, MAXSUCC.

Toute variation de 999 à la limite fixée pour les entiers (ceci dépend du centre de calcul) entraîne l'ajout dans le format de sortie de la procédure IMPRIMER d'une position complémentaire par puissance de 10. (i.e. à la place de - ZZD ($\lceil \mathrm{Log}_{10} N \rceil - 1$)Z D).

Toute variation au-delà de la limite fixée pour les entiers entraîn le travail sur deux éléments et donc une modification totale du programme. De toutes façons, avant d'atteindre cette limite, la limite de la mémoire centra sera dépassée, ce qui oblige à utiliser la seconde version de ce programme av pile externe.

LISTE D'ENCOMBREMENT STATIQUE.

```
UEBERSETZE
 1 QUELLE          = ERDUM
 2 SPRACHE         = ALGGC
 3 NUMERIERUNG     = -STD-
 4 MD              = -STD-
 5 VARIANTE        = D
 6 PROTOKOLL       = AUS
 7 OV              = -STD-

START PSGALGOLCOMP (35,00)
 BEFEHLE          295  GANZWORTE
 VARIABLEN        142  GANZWORTE
 KONSTANTEN        29  GANZWORTE
    COMPIT              BEFEHLE     158  GW  ARBEITSSPEICHER    25  GW
    COMPIT              BEFEHLE     207  GW  ARBEITSSPEICHER    32  GW
    SUP                 BEFEHLE      20  GW  ARBEITSSPEICHER    17  GW
    MODEND              BEFEHLE      53  GW  ARBEITSSPEICHER    19  GW
    FLIMIN              BEFEHLE     123  GW  ARBEITSSPEICHER    23  GW
    DISCAR              BEFEHLE      71  GW  ARBEITSSPEICHER    22  GW
    CHKISI              BEFEHLE      19  GW  ARBEITSSPEICHER    16  GW
    FLIMIN              BEFEHLE      68  GW  ARBEITSSPEICHER    16  GW
    SCHEDU              BEFEHLE     271  GW  ARBEITSSPEICHER    31  GW
    AVOBII              BEFEHLE      44  GW  ARBEITSSPEICHER    21  GW
    TIMRIN              BEFEHLE      58  GW  ARBEITSSPEICHER    22  GW
    SCHEDU              BEFEHLE      72  GW  ARBEITSSPEICHER    18  GW
MIT STOP WURDE ERZEUGT
```

Befehle : code

Variablen : variables

Konstanten : constantes

Arbeitsspeicher : zones de travail

Ganzworte ou GW : mots-machine TR-440 de 48 bits.

Soit au total..... : 1453 Mots pour le code,

29 Mots pour les constantes,

409 Mots pour les zones de travail ;

auxquels il faut ajouter : 20K Mots pour les procédures du système ALGOL,

et..... : $5|\mathcal{E}|^2$ Mots pour les variables dynamiques (matrices)

TEXTE SOURCE

```
000010  'BEGIN'
000020C 'COMMENT' THIS PROGRAM CALLED ORDON
000030C COMPUTES A SCHEDULE FOR NON PREEMPTIVE
000040C RESOURCES. THERE ARE N TASKS AND C TYPES
000050C OF RESOURCES. FOR EACH TYPE J THERE ARE MJ
000060C RESOURCES OF THIS TYPE. THE TIME
000070C CONSTRAINTS OF EACH TASK ARE DEFINED BY AN
000080C ARRIVAL TIME, A DURATION AND A DEADLINE.
000090C WITH EACH TASK AND EACH TYPE OF RESOURCES
000100C IS ASSOCIATED THE NUMBER OF RESOURCES OF
000110C THIS TYPE THE TASK NEEDS.
000120C THE TASKS ARE RELATED BY PRECEDENCE RELATIONS;
000130

000140
000150C 'COMMENT' DECLARATION AND INITIALIZATION OF THE
000160C ***************************************************
000170C SIZE PARAMETERS
000180C ***************;
000190
000200  'INTEGER' N; 'COMMENT' NUMBER OF TASKS;
000210  'INTEGER' C; 'COMMENT' NUMBER OF RES-TYPES;
000220  'INTEGER' M; 'COMMENT' TOTAL NUMBER OF RES;
000230  'INTEGER' MAXSUCC; 'COMMENT' MAXIMUM NUMBER OF SUCCESSORS FOR A TA
000240
000250  'INTEGER' INPLT; 'COMMENT' INPUT DEVICE;
000260
000270  INPUT:=1; SYSUNIT(1,INPUT);
000280  READ (N,C,M,MAXSUCC);
000290
```

```
00030
00031C 'BEGIN' 'COMMENT' DECLARATION OF THE DATA
00032C ************************************
00033C                   THAT DESCRIBE THE PROBLEM
00034C                   *************************;
00035
00036
00037   'INTEGER' 'ARRAY' NBRESOURCESTYPE [1:C];
00038C  'COMMENT' FOR EACH TYPE, THE NB. OF RES. IT CONTAINS;
00039
00040   'INTEGER' 'ARRAY' ARRIVALTIME,DURATION,DEADLINE[0:N];
00041C 'COMMENT' TIME CONSTRAINTS. NOTE THAT ARRIVALTIME COULD
00042C           BE DISCARDED. ITS ONLY USE IS TO INITIATE THE
00043C           COMPUTATION VARIABLE SIGMAY, AND ITS VALUE IS
00044C           KEEPED UNCHANGED IN SIGMAY[*,0];
00045
00046   'INTEGER' 'ARRAY' WISHES [0:N,1:C];
00047C 'COMMENT' FOR EACH TASK T, FOR EACH TYPE J
00048C           WISHES [T,J] CONTAINS THE NUMBER OF
00049C           RES. OF THIS TYPE THE TASK NEEDS;
00050
00051   'INTEGER' 'ARRAY' SUCCESSORS [0:N,1:MAXSUCC+1];
00052C 'COMMENT' FOR EACH TASK T SUCCESSORS [T,*]
00053C           CONTAINS THE LIST OF ITS SUCCESSORS
00054C           FOLLOWED BY THE MARK -1;
00055
00056   'INTEGER' 'ARRAY' NBPREDECESSORS [0:N];
00057C 'COMMENT' FOR EACH TASK T NBPRED [T] CONTAINS
00058C           THE NUMBER OF ITS DIRECT PREDECES
00059C           NOTE THAT THIS ARRAY COULD BE DISCARDED
00060C           IT IS USED ONCE TO INITIATE THE COMP. VAR.
00061C           REM. PRED. AND ITS VALUE IS KEEPED UNCHANGED
00062C           IN REM. PRED. [*,0];
00063
00064C 'COMMENT' THESE TWO ARRAYS ARE ONE OF THE
00065C           POSSIBLE REPRESENTATIONS FOR
00066C           PRECEDENCE RELATIONS. IT MAY BE
00067C           CHANGED. THEN THE PROCEDURES
00068C           COMPUTEAVOFY, COMPUTESIGMAY, MUST
00069C           BE CHANGED TOO;
00070
```

```
00071?
00072?C 'COMMENT' DECLARATION OF THE CROON'S COMPUTATION
00073?C ***************************************************
00074?C             VARIABLES.
00075?C             **********
00076?C
00077?C             THESE VARIABLES ARE THEORICALLY
00078?C             ARRAYS, BUT FOR THE I+1TH STEP OF THE
00079?C             COMPUTATION, ONLY THE ITH COLUMN IS
00080?C             NEEDED, THEREFORE, EVEN IF THIS FIRST
00081?C             PROGRAM USES AN ARRAY REPRESENTATION
00082?C             THE I-1 FIRST COLUMNS MAY BE KEEPED
00083?C             IN A STACK STORED ON A DISK OR ON A
00084?C             TAPE WITH BACK LECTURE;
00085?
00086?   'INTEGER' 'ARRAY' Y[0:N];
00087?C 'COMMENT' CONTAINS THE ACTUALLY CONSIDERED
00088?C             PERMUTATION, THAT IS Y[I] IS THE TASK WHICH
00089?C             IS SCHEDULED AS ITH IN THE CONSIDERED SCHEDULE;
00090?
00091?   'INTEGER' 'ARRAY' AVOFY [0:N,1:N+1];
00092?C 'COMMENT' FOR EACH STEP I, AVOFY[I] CONTAINS
00093?C             THE LIST OF ALL THE TASKS THAT CAN
00094?C             BE SCHEDULED AT THIS STEP ACCORDING
00095?C             TO PRECEDENCE RELATIONS, THAT IS
00096?C             AT MOST N INTEGERS FOR THE TASKS'
00097?C             NAMES AND THE END-OF-LIST MARK -1;
00098?
00099?   'INTEGER' 'ARRAY' AV [0:N,1:N+1];
00100?C 'COMMENT' FOR EACH STEP I, AV[I] CONTAINS THE
00101?C             LIST OF ALL THE TASKS THAT BELONG
00102?C             TO AVOFY[I], COULD BE TRIED
00103?C             (INTEREST CONSTRAINTS) AND HAVE NOT
00104?C             BEEN TRIED. THAT IS, AT MOST N
00105?C             INTEGERS AND THE MARK -1;
00106?
00107?   'INTEGER' 'ARRAY' LIBY[1:M,0:N];
00108?C 'COMMENT' FOR EACH RES. TYPE J, MJ SEQUENTIAL
00109?C             ELEMENTS OF THIS ARRAY ARE RESERVED
00110?C             (TOTAL NUMBER M) TO STORE FOR EACH
00111?C             RESOURCE OF THIS TYPE THE DATE AT
00112?C             WHICH IT WOULD BE FREE, CONSIDERING
00113?C             THAT THE TASKS ARE SCHEDULED ACCOR-
00114?C             DING TO THE ACTUAL PERMUTATION Y AND
00115?C             TO THE SHP POLICY. THE LIBERATION
00116?C             DATES ARE SUPPOSED TO BE STORED
00117?C             ACCORDING TO INCREASING VALUES ORDER
00118?C             IF THESE MJ ELEMENTS RESERVED FOR J;
00119?
```

```
00120°C 'COMMENT'
00121°C 'INTEGER' 'ARRAY' PLUSTOTY [0:N,0:N];
00122°C 'COMMENT' FOR EACH TASK T, FOR EACH STEP I
00123°C            PLUTOTY [T,I] CONTAINS THE DATE AT
00124°C            WHICH T COULD BE SCHEDULED ACCORDING
00125°C            TO PRECEDENCE RELATIONS, ARRIVAL TIME,
00126°C            Y AND THE SHP. THIS VALUE IS ONLY
00127°C            INTERESTING FOR THE TASKS THAT HAVE
00128°C            NOT BEEN SCHEDULED YET, AND THAT
00129°C            CANNOT BE SCHEDULED ACCORDING TO PRECEDENCE
00130°C            RELATIONS, FOR THE ALREADY SCHEDULED
00131°C            ONES, ONLY THE ACTUAL ACTIVATION DATE
00132°C            IS INTERESTING, FOR THESE THAT COULD
00133°C            BE TRIED, ONLY THE SUP OF PLUS TOT Y
00134°C            AND OF THE RESOURCE CONSTRAINTS IS
00135°C            INTERESTING (SIGMA-Y). THESE 3 VALUES
00136°C            NEVER EXIST AT THE SAME TIME.
00137°C            THEREFORE, PLUS TOT-Y WILL BE STORED
00138°C            IN THE SAME ARRAY THAN ACTUAL START
00139°C            TIME AND POSSIBLE START TIME
00140°C            THAT IS , THERE WILL BE ONE ARRAY
00141°C            SIGMAY WHERE SIGMAY[T,I] IS:
00142°C            -FOR THE TASKS THAT ARE SCHEDULED,
00143°C             THEIR ACTUAL START TIME.
00144°C            -FOR THE TASKS THAT BELONG TO
00145°C             AVEFY[I,X] THEIR POSSIBLE START TIME
00146°C             IF SCHEDULED AT THIS STEP.
00147°C            -FOR THE OTHER ONES, THEIR EARLIEST
00148°C             START TIME ACCORDING TO TIME AND
00149°C             PRECEDENCE CONSTRAINTS;
00150°
00151°  'INTEGER' 'ARRAY' SIGMAY [0:N,0:N];
00152°  'COMMENT' CF. PLUS TOT Y;
00153°
00154°  'INTEGER' 'ARRAY' REMAININGPREDECESSORS [0:N,0:N];
00155°C 'COMMENT' FOR EACH TASK T, FOR EACH STEP I,
00156°C            REM.PRED.[T,I] CONTAINS THE NUMBER OF
00157°C            DIRECT PREDECESSORS OF T THAT HAVE
00158°C            NOT BEEN SCHEDULED YET;
00159°
```

```
00160
00161C 'COMMENT' PROGRAMMATION VARIABLES
00162C ********************************;
00163
00164
00165   'INTEGER' I;
00166   'COMMENT' LEVEL COUNTER;
00167
00168   'INTEGER' 'ARRAY' DEBUTTYPE [1:C];
00169C 'COMMENT' THIS ARRAY CONTAINS FOR EACH TYPE THE
00170C           INDEX OF THE ELEMENT OF LIBY RELATED
00171C           TO THE FIRST RES. OF THIS TYPE. RECALL
00172C           THAT RESOURCES OF THE SAME TYPE ARE
00173C           STORED ACCORDING TO LIB. DATES IN
00174C           SEQUENTIAL ELEMENTS OF LIBY;
00175
00176   'INTEGER' 'ARRAY' LENGTHOFAV [0:N];
00177C 'COMMENT' FOR EACH STEP I, LENGTHOFAV[I]
00178C           CONTAINS THE NUMBER OF REMAINING
00179C           TASKS. NOTE THAT AT THE BEGINNING OF
00180C           THE STEP IT IS USED TO BUILD AV[I];
00181
00182   'INTEGER' DUMMY;
00183   'COMMENT' USED FOR WHILE LOOPS;
00184
00185   'INTEGER' INDEX1, INDEX2, SUCCESSOR, DEBUT;
00186
```

```
00187C
00188C 'COMMENT' DECLARATION OF THE PROCEDURES
00189C ************************************
00190C
00191C
00192C NOTE THAT THESE PROCEDURES HAVE NO EXPLICIT
00193C PARAMETERS, THE IMPLICIT PARAMETERS ARE I AND
00194C THE I-1TH COLUMN OF EACH ARRAY. THE RESULT IS
00195C THE ITH COLUMN OF AN ARRAY.;
00196C
```

```
00197C
00198C   'PROCEDURE' COMPUTEAVOFY;
00199C   'BEGIN'
00200CC  'COMMENT' THIS PROCEDURE COMPUTES THE VALUE OF
00201CC            THE CONSTANT (FOR A GIVEN PROBLEM)
00202CC            AV[I,Y] IF USING THE THEORITICAL
00203CC            NOTATION, OR AVOFY[I], IF USING THE
00204CC            PROGRAM NOTATION, Y BEING THE
00205CC            ACTUALLY CONSIDERED PERMUTATION
00206CC            AND I THE ACTUALLY CONSIDERED STEP
00207CC            THIS VALUE IS GIVEN BY:
00208CC            AV[I,Y]=AV[I-1,Y]-Y[I-1]+
00209CC                    (T IS SUCC. Y[I-1],REM. PRED.[T,I]=0);
00210C
00211C   'INTEGER' INDEX1, LENGTH, ELEM, SUCC, IMINUSONE, YOFIMINUSONE;
00212C
00213C   IMINUSONE:=I-1;
00214C   YOFIMINUSONE:=Y[IMINUSONE];
00215C
00216C   'COMMENT' COMPUTATION OF AV[I,Y]=AV[I-1,Y]-Y[I-1];
00217C   INDEX1:=1; LENGTH:=0;
00218C   ELEM:=AVOFY[IMINUSONE,1];
00219C   'FOR' DUMMY:= 0 'WHILE' ELEM ! -1 'DO'
00220C    'BEGIN' 'IF' ELEM ! YOFIMINUSONE
00221C            'THEN' 'BEGIN' LENGTH:=LENGTH+1;
00222C                     AVOFY[I,LENGTH]:=ELEM;
00223C                   'END';
00224C            INDEX1:=INDEX1+1;
00225C            ELEM:=AVOFY[IMINUSONE,INDEX1];
00226C    'END' FIRST TERM;
00227C
00228CC  'COMMENT' COMPUTION OF THE SECOND TERM, RECALL
00229CC            THAT REM,PRED,[T,I]=REM,PRED,[T,I-1]
00230CC            IF T IS NOT A SUCC OF Y[I-1], AND
00231CC            REM,PRED,[T,I-1]-1 IF T IS;
00232C
00233C   'FOR' INDEX1:=0 'STEP' 1 'UNTIL' N 'DO'
00234C         REMAININGPREDECESSORS[INDEX1,I]:=
00235C         REMAININGPREDECESSORS[INDEX1,IMINUSONE];
00236C
00237C   INDEX1:=1; SUCC:= SUCCESSORS[YOFIMINUSONE,1];
00238C   'FOR' DUMMY:=0 'WHILE' SUCC ! -1 'DO'
00239C    'BEGIN' REMAININGPREDECESSORS[SUCC,I]:=
00240C            REMAININGPREDECESSORS[SUCC,I]-1;
00241C            'IF' REMAININGPREDECESSORS[SUCC,I]=0
00242C            'THEN' 'BEGIN' LENGTH:=LENGTH+1;
00243C                           AVOFY[I,LENGTH]:=SUCC;
00244C                   'END';
00245C            INDEX1:=INDEX1+1;
00246C            SUCC:=SUCCESSORS[YOFIMINUSONE,INDEX1];
00247C    'END';
00248C
00249C   LENGTHOFAV[I]:=LENGTH;
00250C   AVOFY[I,LENGTH+1]:=-1; 'COMMENT' END OF LIST;
00251C
00252C   'END' COMPUTEAVOFY;
```

```
00253^
00254^  'COMMENT' ************************************;
00255^
00256^  'PROCEDURE' COMPUTESIGMAY;
00257^  'BEGIN'
00258^C 'COMMENT' RECALL THAT FOR EACH TASK T, FOR
00259^C           EACH  STEP I SIGMAY [T,I] CONTAINS
00260^C           ONE OF THE THREE FOLLOWING POSSIBLE
00261^C           VALLES:
00262^C           1-IF T HAS ALREADY BEEN SCHEDULED,ITS
00263^C             ACTUAL START TIME
00264^C           2-IF T BELONGS TO AVOFY[I] THEN ITS
00265^C             POSSIBLE START TIME IF SCHEDULED AT
00266^C             THIS STEP ACCORDING TO Y, THE SHP
00267^C             AND THE RES. & PREC. CONSTRAINTS
00268^C           3-ELSE, THE DATE AT WHICH T MAY BE
00269^C             SCHEDULED ACCORDING TO ITS ARRIVAL
00270^C             TIME AND TO ITS ALREADY SCHEDULED
00271^C             PREDECESSORS
00272^C           THEREFORE:
00273^C           IF T FULFILLS 1 THEN SIG[T,I]=SIG[T,I-1]
00274^C           IF T FULFILLS 2, THEN SIG[T,I]=
00275^C           SUP( SIG[T,I-1],NEW RES. LIB. DATES)
00276^C           IF T FULFILLS 3, THEN IF T IS A SUCC
00277^C           OF Y[I-1], THEN SIG[T,I]=
00278^C           SUP(SIG[T,I-1],FINISH.TIME[Y[I-1]])
00279^C           ELSE, SIG[T,I]=SIG[T,I-1];
00280^
00281^
00282^  'INTEGER' SUCC, IMINUSONE, YOFIMINUSONE, T, N, WITHFREE,
00283^           PLUSTOTMACHINES, INDEX1, INDEX2, ENDOFYOFIMINUSONE;
00284^
00285^
00286^  IMINUSONE:=I-1;
00287^  YOFIMINUSONE:=Y[IMINUSONE];
00288^  ENDOFYOFIMINUSONE:=SIGMAY[YOFIMINUSONE,IMINUSONE]
00289^                    +DURATION[YOFIMINUSONE];
00290^
00291^
00292^C 'COMMENT' IT IS EASIER AND CHEAPER TO CONSIDER
00293^C           FIRST THAT SIG[T,I]=SIG[T,I-1] FOR
00294^C           EACH TASK AND TO COMPUTE A NEW VALUE
00295^C           WHEN NOT THAN TO COMPUTE ALL THE
00296^C           DIFFERENT CASES SEPARATLY;
00297^
00298^  'FOR' T:=0 'STEP' 1 'UNTIL' N 'DO'
00299^        SIGMAY[T,I]:=SIGMAY[T,IMINUSONE];
```

```
00300
00301C 'COMMENT' LET US COMPUTE NOW THE VALUE OF SIG.
00302C            FOR THE SUCCESSORS OF V[I-1] USING
00303C            TWO ARRAYS, THIS VALUE WOULD BE PLUSTOTV[T,I];
00304
00305  INDEX1:=1;
00306  SUCC:=SUCCESSORS[VOFIMINUSONE,1];
00307  'FOR' DUMMY:=0 'WHILE' SUCC ! -1 'DO'
00308   'BEGIN'
00309            SIGMAY[SUCC,I]:=SUP(SIGMAY[SUCC,I],
00310                                ENDOFVOFIMINUSONE);
00311            INDEX1:=INDEX1+1;
00312            SUCC:=SUCCESSORS[VOFIMINUSONE,INDEX1];
00313   'END';
00314
00315C 'COMMENT' COMPUTATION OF THE EARLIEST START TIME
00316C            ACCORDING TO RES. CONSTRAINTS FOR THE
00317C            TASKS THAT BELONG TO AVOFV[I];
00318
00319  INDEX1:=1;
00320  T:=AVOFV[I,1];
00321  'FOR' DUMMY:=0 'WHILE' T ! -1 'DO'
00322   'BEGIN' 'COMMENT' FOR EACH TASK THAT BELONG TO AVOFV[I];
00323            PLUSTOTMACHINES:=0;
00324            'FOR' INDEX2:=1 'STEP' 1 'UNTIL' C 'DO'
00325             'BEGIN' 'COMMENT' FOR EACH RES.TYPE;
00326                     W:=WISHES[T,INDEX2];
00327                     'IF' W ! 0 'THEN'
00328                      'BEGIN' 'COMMENT' SEE (1);
00329                              WTHERE:=LIBY[DEB'TTYPE[INDEX2]+W-1,
00330                                           INMINUSONE];
00331                              PLUSTOTMACHINES:=
00332                               SUP(PLUSTOTMACHINES,WTHERE);
00333                      'END' IF W NOT 0;
00334
00335             'END' FOR EACH TYPE;
00336
00337             SIGMAY[T,I]:=SUP(PLUSTOTMACHINES,SIGMAY[T,I]);
00338             'COMMENT' SEE (1,1);
00339             INDEX1:=INDEX1+1;
00340             T:=AVOFV[I,INDEX1];
00341   'END' FOR EACH TASK THAT BELONG TO AVOFV[I];
00342
00343  'END' COMPUTESIGMAY;
```

```
003440
003450  'COMMENT' **********************************;
003460
003470  'INTEGER' 'PROCEDURE' SUP(A,B);
003480  'VALUE' A,B;
003490  'INTEGER' A,B;
003500  'BEGIN'
003510
003520  SUP:= 'IF' A >= B 'THEN' A 'ELSE' B;
003530
003540  'END' SUP;
003550
```

```
00356   'COMMENT' ************************************;
00357
00358   'BOOLEAN' 'PROCEDURE' NODEADLINE;
00359   'BEGIN'
00360C  'COMMENT' FOR FACILITY'S SAKE, THE ARRAY DEADLINE
00361C            IS NOT SET TO DEADLINE VALUES,BUT TO
00362C            DEADLINE-DURATION. THEREFORE, A SIMPLE
00363C            COMPARISON IS SUFFICIENT.
00364C            AS BEFORE, IT IS CHEAPER TO DO THIS
00365C            COMPARISON FOR ALL THE TASKS, INCLUDING
00366C            THOSE FOR WHICH SIG. IS THE SAME AS
00367C            FOR THE PREC. STEP THAT TO TRY TO
00368C            RECOGNIZE THEM.
00369C            TO MAKE THE LECTURE EASIER, COMPUTESIG.
00370C            AND NODEADLINE HAVE BEEN SEPARATED,
00371C            BUT IN AN OPTIMIZED RELEASE, IT
00372C            WOULD BE BETTER TO DO THE COMPARISON
00373C            WHEN COMPUTING SIGMAY.;
00374
00375   'INTEGER' T;
00376
00377   T:=1;
00378   'FOR' DUMMY:=0 'WHILE' ((T<N)
00379                           'AND'
00380                           (SIGMAY[T,I]<=DEADLINE[T])) 'DO'
00381    T:=T+1;
00382
00383C 'COMMENT' AT THIS POINT, WHETHER T<N, THAT WOULD
00384C            MEAN THAT A  DEADLINE HAS BEEN
00385C            PASSED, OR T=N AND THE LAST COMPARISON
00386C            HAS NOT BEEN DONE;
00387
00388   NODEADLINE:= (T = N) 'AND' (SIGMAY[N,I] <= DEADLINE[N]);
00389
00390   'END' NODEADLINE;
00391
```

```
003920  'COMMENT' **********************************;
003930
003940  'PROCEDURE' ELIMINATION1;
003950  'BEGIN'
003960C 'COMMENT' THIS PROCEDURE APPLIES THEOREM 1, THAT IS
003970C           IF IN AV[I,Y], THERE ARE TWO TASKS T1,T2 SUCH THAT:
003980C           SIG[T1,I]+DURATION[T1]<=SIG[T2,I]
003990C           THEN T2 MUST BE DISCARDED FROM AV[I,Y].
004000C           THE ALGORITHM IS THE FOLLOWING:
004010C           -A TASK T1 IS COMPARED IN BOTH WAYS
004020C            WITH ALL ITS FOLLOWERS IN AV[I,Y].
004030C           -IF T1 MAKES POSSIBLE THE DISCARDING OF
004040C            ANY T2, THEN T2 IS DISCARDED AND THE
004050C            ALGORITHM GOES ON WITH T1 AND THE
004060C            FOLLOWERS OF T2.
004070C           -IF ANY T2 MAKES POSSIBLE THE DISCARDING
004080C            OF T1, THEN T1 IS DISCARDED AND THE
004090C            ALGORITHM RESTARTS WITH THE FOLLOWER OF T1.
004100C           -IF NO T IS DISCARDED, THEN THE ALGORITHM
004110C            RESTARTS WITH THE FOLLOWER OF T1.;
004120
004130  'INTEGER' INDEX1, INDEX2, T1, T2, POSSENDT1, POSSENDT2,
004140            SIGT1I, SIGT2I;
004150  'BOOLEAN' DISCARDT1;
004160
```

```
00417    INDEX1:=1; T1:=AV[I,1];
00418    'COMMENT' FOR EACH T1 THAT BELONG TO AV[I];
00419    'FOR' DUMMY:=0 'WHILE' T1 ! =-1 'DO'
00420     'BEGIN' SIGT1I:=SIGMAY[T1,I];
00421             POSSENDT1:=SIGT1I+DURATION[T1];
00422             DISCARDT1:='FALSE';
00423             INDEX2:=INDEX1+1;
00424             T2:=AV[I,INDEX2];
00425             'COMMENT' FOR EACH T2 FOLLOWER OF T1;
00426             'FOR' DUMMY:=0 'WHILE' (T2 ! =-1) 'AND'(¬ DISCARDT1) 'DO'
00427              'BEGIN' SIGT2I:=SIGMAY[T2,I];
00428                      POSSENDT2:=SIGT2I+DURATION[T2];
00429                      'IF' POSSENDT1 <= SIGT2I 'THEN'
00430                       'BEGIN' DISCARDFROMAVOFI(T2);
00431                               T2:=AV[I,INDEX2];
00432                               'COMMENT' SEE (2);
00433C                       'END'
00434C                      'ELSE'
00435                       'IF' POSSENDT2>SIGT1I 'THEN'
00436                        'BEGIN' INDEX2:=INDEX2+1;
00437                                T2:=AV[I,INDEX2];
00438                                'COMMENT' SEE (3);
00439C                       'END'
00440C                      'ELSE'
00441                       'BEGIN' DISCARDFROMAVOFI(T1);
00442                               DISCARDT1:='TRUE';
00443                               'COMMENT' SEE (4);
00444                       'END' TASK T2;
00445              'END' FOLLOWERS OF T1;
00446C             'COMMENT' AT THIS POINT, WHETHER T2=-1
00447C                       THAT IS T1 HAS BEEN COMPARED
00448C                       SUCCESSFULLY WITH ALL ITS
00449C                       FOLLOWERS, PERHAPS DISCARDING
00450C                       SOME OF THEM, OR DISCARDT1=1
00451C                       THAT IS T1 HAS BEEN DISCARDED,
00452C                       IN BOTH CASES, THE ALGORITHM
00453C                       RESTARTS WITH THE FOLLOWER
00454C                       OF T1, THAN CAN OCCUPY THE SAME
00455C                       PLACE, IF T1 HAS BEEN DISCARDED
00456C                       THE WHILE LOOP WILL TEST IF
00457C                       THERE IS A FOLLOWER AT ALL.;
00458              'IF' T2=-1 'THEN' INDEX1:=INDEX1+1;
00459              T1:=AV[I,INDEX1];
00460     'END' FOR EACH TASK T1;
00461C  'COMMENT' AT THIS POINT, ALL THE TASKS HAVE BEEN
00462C             COMPARED TWO BY TWO, AND THOSE THAT
00463C             HAD TO BE ELIMINATED HAVE BEEN;
00464
00465    'END' ELIMINATION1;
00466
```

```
'COMMENT' ************************************;

'PROCEDURE' DISCARDFROMAVOFI (T);
'VALUE' T;
'INTEGER' T;
'BEGIN'
'COMMENT' THIS PROCEDURE DISCARDS THE TASK WHICH
          NUMBER IS GIVEN AS PARAMETER FROM THE
          LIST AV[I], I IS NO PARAMETER, BUT
          THE CURRENT VALUE OF THE LEVEL COUNTER,
          THE LIST IS THEN COMPACTED. RECALL THAT
          A LIST IS A VECTOR THAT CONTAINS INTEGERS,
          FOLLOWED BY THE MARK -1, AND WHICH
          LENGTH IS KEEPED IN LENGTHOFAVOFI[I];

'INTEGER' INDEX1,LENGTH;
LENGTH:=LENGTHOFAV[I]; INDEX1:=1;
'FOR' DUMMY:=0 'WHILE' (INDEX1<=LENGTH) 'AND'
                       (AV[I,INDEX1] ¬= T)
'DO'  INDEX1:=INDEX1+1;
'COMMENT' THIS LOOP ONLY SEARCHES THE PLACE OF THE
          ELEMENT, THEFORE THERE IS NO ACTION;

'FOR' INDEX1:=INDEX1 'STEP' 1 'UNTIL' LENGTH 'DO'
 AV[I,INDEX1]:=AV[I,INDEX1+1];
'COMMENT' THIS LOOP DISPLACES ALL THE FOLLOWERS OF T
          INCLUDING THE MARK -1;
LENGTHOFAV[I]:=LENGTH-1;

'END' DISCARD FROM AV[I];
```

```
00498C   'COMMENT' **********************************;
00499C
00500C   'INTEGER' 'PROCEDURE' CHOISIR;
00501C   'BEGIN'
00502CC  'COMMENT' THIS PROCEDURE CHOOSES ONE TASK FROM
00503CC             AV[I], HAVING NO RESULT ABOUT THE
00504CC             DIFFERENT CHOICE LAWS, WE USE THERE A FIFO;
00505C   CHOISIR:=AV[I,1];
00506C
00507C   'END' CHOISIR;
00508C
```

```
00509C  'COMMENT' ************************************;
00510C
00511C  'BOOLEAN' 'PROCEDURE' ELIMINATION2;
00512C  'BEGIN'
00513CC 'COMMENT' THIS PROCEDURE APPLIES THEOREM 2, THAT
00514CC           IS, IF SIG[Y[I],I]<SIG[Y[I-1],I-1]
00515CC                  OR
00516CC                  (SIG[Y[I],I]=SIG[Y[I-1],I-1]
00517CC                   AND Y[I]<Y[I-1])
00518CC                THEN IT IS USELESS TO GO ON;
00519C  ELIMINATION2:= SIGMAY[Y[I],I]<SIGMAY[Y[I-1],I-1]
00520C                'OR'
00521C                 (SIGMAY[Y[I],I]=SIGMAY[Y[I-1],I-1]
00522C                  'AND' Y[I]<Y[I-1]);
00523C  'END' ELIMINATION2;
00524C
```

```
'COMMENT' ************************************;

'PROCEDURE' SCHEDULE (T);
'VALUE' T;
'INTEGER' T;
'BEGIN'
'COMMENT' THIS PROCEDURE ALLOCATES ITS RESOURCES TO
          THE CHOOSEN Y[I], AS WE DON'T INTEND TO
          DRAW THE SCHEDULE, NO READABLE TRACE
          WILL BE KEEPED, WE ONLY MODIFY SOME VALUES
          IN LIBY (ACCORDING TO THE NEEDS OF T AND TO
          THE SHP), AND REORDER THE ARRAY, IT IS
          OF COURSE POSSIBLE TO KEEP TRACK ANYWHERE
          OF THE NAMES OF THE ALLOCATED RES.
          RECALL THE PROPERTIES OF LIBY AND DEBUTTYPE;

'INTEGER' TYPE, RESOURCE, LIBRESOURCE, ENDTYPE,
          STARTTIMEOFT, ENDOFT, NOMOVE, INMINUSONE,
          W, REORDERED;
```

```
05440   STARTTIMEOFT:=SIGMAY[T,I]; ENDOFT:=STARTTIMEOFT+DURATION[T];
05450   IMINUSONE:=I-1;
05460   'FOR' TYPE:=1 'STEP' 1 'UNTIL' C 'DO'
05470    'BEGIN' 'COMMENT' FOR EACH TYPE;
05480            W:=WISHES[T,TYPE];
05490            'IF' W ! 0 'THEN'
05500             'BEGIN' ENDTYPE:=DEBUTTYPE[TYPE]+NBRESOURCESTYPE[TYPE]-1;
05510                     RESOURCE:=ENDTYPE;
05520                     NOMOVE:=ENDTYPE+1;
05530                     LIBRESOURCE:=LIBY[RESOURCE,IMINUSONE];
05540                     'FOR' DUMMY:=0 'WHILE' LIBRESOURCE>STARTTIMEOFT 'D
05550                      'BEGIN' 'IF' LIBRESOURCE>ENDOFT 'THEN' NOMOVE:=NO
05560                              RESOURCE:=RESOURCE-1;
05570                              LIBRESOURCE:=LIBY[RESOURCE,IMINUSONE];
05580                      'END';
05590                      'COMMENT' SEE (6);
05600                     INDEX2:=RESOURCE-W; 'COMMENT' SEE (7);
05610                     'IF' INDEX2 >= DEBUTTYPE[TYPE] 'THEN'
05620                       'BEGIN' 'COMMENT' SEE (8);
05630                         'FOR' INDEX1:=DEBUTTYPE[TYPE]
05640                         'STEP' 1 'UNTIL' INDEX2 'DO'
05650                           LIBY[INDEX1,I]:=LIBY[INDEX1,IMINUSONE];
05660                       'END';
05670                     'IF' NOMOVE <= ENDTYPE 'THEN'
05680                       'BEGIN' 'COMMENT' SEE (9);
05690                         'FOR' INDEX1:=NOMOVE
05700                         'STEP' 1 'UNTIL' ENDTYPE 'DO'
05710                           LIBY[INDEX1,I]:=LIBY[INDEX1,IMINUSONE];
05720                       'END';
05730                     INDEX1:=RESOURCE-W+1;
05740                     'COMMENT' INDEX OF THE FIRST ALL. RES.;
05750                     INDEX2:=RESOURCE+1;
05760                     'COMMENT' INDEX OF THE FIRST DISPLACED RES.;
05770                     'FOR' REORDERED:=NOMOVE-RESOURCE-1
05780                     'STEP' -1 'UNTIL' 1 'DO'
05790                       'BEGIN' LIBY[INDEX1,I]:=LIBY[INDEX2,IMINUSONE];
05800                               INDEX1:=INDEX1+1;
05810                               INDEX2:=INDEX2+1;
05820                       'END';
05830                     'COMMENT' SEE (10);
05840                     INDEX2:=NOMOVE-1;
05850                     'FOR' INDEX1:=NOMOVE-W 'STEP' 1 'UNTIL' INDEX2 'DO'
05860                      LIBY[INDEX1,I]:=ENDOFT;
05870            'END' IF W > 0
05880           'ELSE'
05890            'BEGIN' ENDTYPE:=DEBUTTYPE[TYPE]+NBRESOURCESTYPE[TYPE]-1;
05900             'FOR' RESOURCE:=DEBUTTYPE[TYPE]
05910             'STEP' 1 'UNTIL' ENDTYPE 'DO'
05920              LIBY[RESOURCE,I]:=LIBY[RESOURCE,IMINUSONE];
05930            'END' ELSE W = 0 END IF;
05940    'END' TYPE;
05950  'END' SCHEDULE;
05960
```

```
00597C  'COMMENT' ************************************;
00598C
00599C  'PROCEDURE' AVOFIISSETTOAVOFY;
00600C  'BEGIN'
00601CC 'COMMENT' THE AIM OF THIS PROCEDURE IS TO INITIALIZE
00602CC           THE VARIABLE AV[I] WITH THE VALUE OF THE
00603CC           PROBLEM'S CONSTANT AV[I,Y]. RECALL THAT
00604CC           AV[I,Y] OR AVOFY[I] IS A PROBLEM'S CONSTANT
00605CC           EVEN IF IT IS COMPUTED, FOR Y AND I FIXED,
00606CC           ITS VALUE DOES NOT DEPEND ON TIME. ON THE
00607CC           OTHER HAND, AV[I] IS A PROGRAM'S VARIABLE
00608CC           THAT IS, ITS VALUE IS TIME DEPENDING,
00609CC           IT CONTAINS THE TASKS THAT HAVE NOT
00610CC           BEEN TRIED;
00611C
00612C  'INTEGER' INDEX1,LENGTH;
00613C
00614C  LENGTH:=LENGTHOFAV[I]+1; 'COMMENT' +1 TO INCLUDE THE MARK;
00615C  'FOR' INDEX1:=1 'STEP' 1 'UNTIL' LENGTH 'DO'
00616C        AV[I,INDEX1]:=AVOFY[I,INDEX1];
00617C
00618C  'END'AVOFIISSETTOAVOFY;
00619C
```

```
006200   'COMMENT' **********************************;
006210
006220   'PROCEDURE' IMPRIMER;
006230   'BEGIN'
006240C  'COMMENT' FOR THIS FIRST RELEASE, THIS PROCEDURE
006250C            IS A VERY SIMPLE ONE, WE ONLY TYPE THE
006260C .          PERMUTATION AND THE START TIMES, IF
006270C            NEEDED (SEE SCHEDULE), THE NAME OF THE
006280C            RES. MAY BE KEEPED AND TYPED TOO, OR A
006290C            GANTT-CHART MAY BE DRAWN;
006300
006310   'INTEGER' INDEX1, T;
006320
006330   OUTPUT(6,'('*,5/,35B,'('SCHEDULE')',5/')');
006340   'FOR' INDEX1:=1 'STEP' 1 'UNTIL' N 'DO'
006350S         OUTPUT(6,'('/,10B,'('AT STEP')',=ZZDB,
006360S                  '('TASK')',-ZZDB,
006370S                  '('HAS BEEN SCHEDULED WITH START-TIME')',
006380S                  -6ZDB,'('.')''')',
006390                   INDEX1,Y[INDEX1],SIGMAY[Y[INDEX1],INDEX1]);
006400
006410   'END' IMPRIMER;
006420
```

```
'COMMENT' ************************************;

'PROCEDURE' SCHEDULINGSTEPI;
'BEGIN'
'COMMENT' THIS IS THE MAIN ALGORITHM, THE OTHER
          PROCEDURES WERE ONLY SOME SECONDARY
          FUNCTIONS WHICH AIM WAS EASY TO UNDERSTAND
          EVEN IF DIFFICULT TO PROGRAM CLEARLY
          BUT TO UNDERSTAND THIS FUNCTION PLEASE SEE
          THE PAPERS ABOUT ORDON;

COMPUTEAVCFY;
COMPUTESIGMAY;
'IF' NODEADLINE 'THEN'
     'BEGIN' AVCFIISSETTOAVCFY;
             ELIMINATION1;
             'FOR' DUMMY:=0 'WHILE' LENGTHOFAV[I]>0 'DO'
              'BEGIN'
                    Y[I]:=CHOISIR;
                    'IF' ¬ ELIMINATION2 'THEN'
                         'BEGIN'
                                SCHEDULE(Y[I]);
                                'IF' I=1 'THEN'
                                     IMPRIMER
                                'ELSE'
                                     'BEGIN' I:=I+1;
                                             SCHEDULINGSTEPI;
                                             I:=I-1;
                                     'END';
                         'END' ELIM2=FALSE;
                    DISCARDFROMAVCFI(Y[I]);
              'END' FOR EACH TASK OF AV[I];
     'END' IF NO DEADLINE OVER PASSED;

'END' SCHEDULING STEP I;
```

```
00679C
00680C 'COMMENT' INITIALIZATIONS
00681C **************************;
00682C
00683C
00684C 'COMMENT' READ THE DEFINITION OF THE PROBLEM
00685C ----------------------------------------;
00686C
00687C  READ(NBRESOURCESTYPE);
00688C 'COMMENT' FOR EACH TYPE (1:C) THE NUMBER OF RES.
00689C            IT CONTAINS IS READ;
00690C
00691C  READ(ARRIVALTIME,DURATION,DEADLINE);
00692C 'COMMENT' FOR EACH TASK (0:N) THE TIME CONSTRAINTS
00693C            ARE READ. BEWARE THE ORDER!!!!!;
00694C
00695C  READ(WISHES);
00696C 'COMMENT' FOR EACH TASK, FOR EACH TYPE THE WISHES
00697C            (0:I,1:C) ARE READ. BEWARE THE ORDER!!!!!;
00698C
00699C  READ(NBPREDECESSORS);
00700C 'COMMENT' FOR TASK, THE NUMBER OF ITS DIRECT PREDECESSORS
00701C            (0:I) IS READ;
00702C
00703C  'COMMENT' READ THE LISTS OF SUCCESSORS;
00704C  'FOR' INDEX1:=0 'STEP' 1 'UNTIL' N 'DO'
00705C        'BEGIN' INDEX2:=0;
00706C                SUCCESSOR:=0;
00707C                'FOR' INDEX2:=INDEX2+1 'WHILE' SUCCESSOR ! =1 'DO'
00708C                      'BEGIN' READ(SUCCESSOR);
00709C                              SUCCESSORS[INDEX1,INDEX2]:=SUCCESSOR;
00710C                      'END' FOR EACH SUCCESSOR;
00711C        'END' FOR EACH TASK;
00712C
```

```
007130C 'COMMENT' INITIALIZATION OF THE PROGRAM VARIABLES
007140C ***********************************************;
007150C
007160C  AVOFY[0,1]:=0;
007170C  AVOFY[0,2]:=-1;
007180C  AV[0,1]:=0;
007190C  AV[0,2]:=-1;
007200C  LENGTHOFAV[0]:=1;
007210C  Y[0]:=0;
007220C  'FOR' INDEX1:=1 'STEP' 1 'UNTIL' M 'DO'
007230C        LIBY[INDEX1,0]:=0;
007240C  'FOR' INDEX1:=0 'STEP' 1 'UNTIL' N 'DO'
007250C        'BEGIN' SIGMAY[INDEX1,0]:=ARRIVALTIME[INDEX1];
007260C                DEADLINE[INDEX1]:=DEADLINE[INDEX1]-DURATION[INDEX1];
007270C                REMAININGPREDECESSORS[INDEX1,0]:=NBPREDECESSORS[INDEX
007280C        'END';
007290C  DEBUT:=DEBUTTYPE[1]:=1;
007300C  'FOR' INDEX1:=2 'STEP' 1 'UNTIL' C 'DO'
007310C        DEBUTTYPE[INDEX1]:=DEBUT:=DEBUT+NBRESOURCESTYPE[INDEX1-1];
007320C
007330C
```

```
007340C 'COMMENT' PROGRAMME PROPREMENT DIT
007350C **********************************;
007360
007370
007380   I:=1;
007390   SCHEDULINGSTEP1;
007400   OUTPUT(6,'('*,/,10B,'('THAT=S ALL FOLKS')!!)');
007410
007420  'END' SIZE DEPENDING BLOCK;
007430C 'END' PROGRAM ORDON
007440C
```

```
00745°C 'COMMENT' FOOT-NOTES
00746°C **********************
00747°C
00748°C [1]
00749°C     THE TASK NEEDS W RESOURCES OF THIS TYPE,
00750°C     CONSIDERING THAT THE LIB-DATES ARE ORDERED,
00751°C     W RES.OF THIS TYPE WILL BE FREE BEFORE THE
00752°C     W'TH LIB-DATE THAT IS STORED IN THE
00753°C     DEBUTTYPE + W - 1 TH ELEMENT OF LIBY.
00754°C
00755°C [1.1]
00756°C     RECALL THAT SIGNAY CONTAINS SUP (PLUTOTPRECEDENCE,
00757°C     PLUTOTHACHINES), THAT AS SOON AS T BELONGS
00758°C     TO AV[I] PLUTOTPRECEDENCE IS KEEPED UNCHANGED,
00759°C     THAT PLUTOTHACHINES IS NON-DECREASING, AND
00760°C     THAT AS LONG AS T DOES NOT BELONG TO AV[I],
00761°C     PLUTOTHACHINE IS USELESS
00762°C
00763°C [2]
00764°C     T2 MUST BE DISCARDED, THEREFORE, THE TASK TO
00765°C     BE CONSIDERED, (ITS FOLLOWER), NOW OCCUPIES
00766°C     ITS PLACE ACCORDING TO THE PROCEDURE DISCARD.
00767°C
00768°C [3]
00769°C     NOTHING TO BE DONE, THE NEXT TASK (FOLLOWER
00770°C     OF T2) OCCUPIES THE NEXT ARRAY ELT.
00771°C
00772°C [4]
00773°C     T1 MUST BE DISCARDED, WE MUST RESTART WITH
00774°C     ITS FOLLOWER, THEREFORE DISCARDT1=TRUE THAT
00775°C     WILL FORCE A NEW START
00776°C
```

```
007770C [6]
007780C      THE ARRAY MUST BE ORDERED ACCORDING TO TYPES
007790C      AND LIB-DATES
007800C      GIVEN AN ORDERED ARRAY, FOR A TYPE C, THERE
007810C      ARE FOUR KINDS OF RES.
007820C      - THOSE THAT WILL BE ALLOCATED, AND WHICH
007830C        LIB-DATE WILL BECOME ENDOFT, AND THEREFORE
007840C        MUST BE REORDERED, THEY ARE PLACED IN THE MIDDLE
007850C        OF THE PART OF THE ARRAY RESERVED FOR THIS TYPE
007860C      - THOSE WHICH LIB DATE IS ALREADY >= ENDOFT
007870C        IT WILL BE KEEPED UNCHANGED, AND HAVE NOT
007880C        TO BE REORDERED,THEY ARE PLACED AT THE END OF THE
007890C        PART OF THE ARRAY RESERVED FOR THIS TYPE,
007900C       -THOSE WHICH LIB-DATE IS TOO SMALL,
007910C        THEREFORE THEY WILL NOT BE ALLOCATED, (SHP),
007920C        THIS LIB-DATE WILL BE KEEPED UNCHANGED
007930C        AND THEY HAVE NOT TO BE REORDERED,
007940C        THEY ARE PLACED AT THE BEGINNING OF THE
007950C        PART OF THE ARRAY RESERVED FOR THIS TYPE
007960C       -THOSE WHICH LIB-DATE IS LARGER THAN
007970C        STARTIME OF T AND SMALLER THAN ENDOFT,
007980C        THIS VALUE WILL BE KEEPED UNCHANGED, BUT
007990C        THEY MUST BE REORDERED, EXCHANGING THEIR
008000C        PLACE WITH THOSE THAT ARE ALLOCATED
008010C        THEY ARE PLACED BETWEEN THE FIRST AND THE
008020C        SECOND ONES.
008030C      AT THIS POINT OF THE PROGRAM, NOMOVE POINTS
008040C      TO THE FIRST RES. OF TYPE 2: LIB>=ENDOFT
008050C      RESOURCE TO THE LAST RES. WITH LIB<=STARTIME OF T
008060C      THAT IS THE LAST TO BE ALLOCATED.
008070C
008080C [7]
008090C      INDEX2 POINTS TO THE LAST TASK OF TYPE 3,
008100C      TOO EARLY FREE
008110C
008120C [8]
008130C      FOR THESE RES. LIB-DATE IS KEEPED UNCHANGED
008140C      THEY WILL NOT BE REORDERED, AS THEY REMAIN
008150C      THE SMALLEST ONES
008160C
008170C [9]
008180C      THESE LIB-DATES ARE NOT MOVED THEY REMAIN
008190C      THE LARGEST ONES
008200C
008210C [10]
008220C      THERE ARE !!REORDERED!! TASKS TO BE REORDERED,
008230C      THE TWO FIRST TO BE REORDERED ARE POINTED TO BY
008240C      INDEX1 AND INDEX2, THESE ARE THE FIRST TO BE
008250C      ALLOCATED (RESOURCE=W+1) AND THE FIRST NOT TO
008260C      BE ALLOCATED (TOO LATE FREE), RESOURCE=1;
END PROTOKOLL
```

A1-12) EXEMPLE D'EXECUTION.

DONNEES.

```
7;
2;
4;
3;

2, 2;

 0,  0,  3,  3,  5,  0,  ,  0;
 0,  3,  2,  2,  5,  1,  ,  4;
 0, 13, 15, 15, 20, 19, 20, 20;

0, 0;
1, 2;
1, 1;
1, 2;
2, 2;
1, 0;
1, 0;
1, 0;

0, 1, 1, 1, 2, 1, 1, 1;

7, 1, 5, -1;
2, 3,    -1;
4,       -1;
4,       -1;
         -1;
6,       -1;
         -1;
         -1;
```

```
□START
 1 PROGRAMM      = [TESTLC
 2 LANG          = -STD-
 5 DATEI         = 1¬DATAC

START [TESTL]

INPUT OF N, NUMBER OF TASKS
N =    +7
INPUT OF C, NUMBER OF RES.TYPES
C =    +2
INPUT OF M, NUMBER OF RES.
M =    +4
INPUT OF MAXSUCC, MAXIMUM NUMBER OF SUCCESSORS
MAXSUCC =    +3

INPUT OF THE RESOURCES PARTITION
THERE ARE    +2 RESOURCES OF TYPE    1
THERE ARE    +2 RESOURCES OF TYPE    2

INPUT AND CONTROL OF THE INTERNAL CONSISTENCY FOR THE ARRIVAL TIMES
THE ARRIVAL TIME OF TASK 0 IS 0
ARRIVAL TIME OF TASK     1 :    +0
ARRIVAL TIME OF TASK     2 :    +3
ARRIVAL TIME OF TASK     3 :    +3
ARRIVAL TIME OF TASK     4 :    +5
ARRIVAL TIME OF TASK     5 :    +0
ARRIVAL TIME OF TASK     6 :    +1
ARRIVAL TIME OF TASK     7 :    +0

INPUT AND CONTROL OF THE INTERNAL CONSISTENCY FOR THE DURATIONS
THE DURATION OF TASK 0 IS 0
DURATION OF TASK     1 :    +3
DURATION OF TASK     2 :    +2
DURATION OF TASK     3 :    +2
DURATION OF TASK     4 :    +5
DURATION OF TASK     5 :    +1
DURATION OF TASK     6 :    +1
DURATION OF TASK     7 :    +4

INPUT AND CONTROL OF THE INTERNAL CONSISTENCY FOR THE DEADLINES
THE DEADLINE OF TASK 0 IS 0
DEADLINE OF TASK     1 :   +13
DEADLINE OF TASK     2 :   +15
DEADLINE OF TASK     3 :   +15
DEADLINE OF TASK     4 :   +20
DEADLINE OF TASK     5 :   +19
DEADLINE OF TASK     6 :   +20
DEADLINE OF TASK     7 :   +20

INPUT AND CONTROL OF WISHES
WISHES OF TASK 0 FOR TYPE    1 :    +0
WISHES OF TASK 0 FOR TYPE    2 :    +0
WISHES OF TASK    1
FOR TYPE    1    +1
FOR TYPE    2    +2
WISHES OF TASK    2
FOR TYPE    1    +1
FOR TYPE    2    +1
WISHES OF TASK    3
FOR TYPE    1    +1
```

```
FOR TYPE     2     +2
WISHES OF TASK      4
FOR TYPE     1     +2
FOR TYPE     2     +2
WISHES OF TASK      5
FOR TYPE     1     +1
FOR TYPE     2     +0
WISHES OF TASK      6
FOR TYPE     1     +1
FOR TYPE     2     +0
WISHES OF TASK      7
FOR TYPE     1     +1
FOR TYPE     2     +0

INPUT AND CONTROL OF NBPRED
THE NUMBER OF PRED. OF TASK 0 IS 0
THE NUMBER OF DIRECT PRED OF TASK     1 IS     +1
THE NUMBER OF DIRECT PRED OF TASK     2 IS     +1
THE NUMBER OF DIRECT PRED OF TASK     3 IS     +1
THE NUMBER OF DIRECT PRED OF TASK     4 IS     +2
THE NUMBER OF DIRECT PRED OF TASK     5 IS     +1
THE NUMBER OF DIRECT PRED OF TASK     6 IS     +1
THE NUMBER OF DIRECT PRED OF TASK     7 IS     +1

INPUT AND CONTROL OF SUCC.
THE SUCCESSORS OF TASK     0 ARE:   +7     +1     +5
THE SUCCESSORS OF TASK     1 ARE:   +2     +3
THE SUCCESSORS OF TASK     2 ARE:   +4
THE SUCCESSORS OF TASK     3 ARE:   +4
THE SUCCESSORS OF TASK     4 ARE:
THE SUCCESSORS OF TASK     5 ARE:   +6
THE SUCCESSORS OF TASK     6 ARE:
THE SUCCESSORS OF TASK     7 ARE:

ENDE DIESTLO   ).NC
```

RESULTATS.

SCHEDULE

AT STEP 1 TASK 1 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 2 TASK 7 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 3 TASK 5 HAS BEEN SCHEDULED WITH START-TIME 3 .
AT STEP 4 TASK 2 HAS BEEN SCHEDULED WITH START-TIME 4 .
AT STEP 5 TASK 6 HAS BEEN SCHEDULED WITH START-TIME 4 .
AT STEP 6 TASK 3 HAS BEEN SCHEDULED WITH START-TIME 6 .
AT STEP 7 TASK 4 HAS BEEN SCHEDULED WITH START-TIME 8 .

SCHEDULE

AT STEP 1 TASK 1 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 2 TASK 7 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 3 TASK 5 HAS BEEN SCHEDULED WITH START-TIME 3 .
AT STEP 4 TASK 3 HAS BEEN SCHEDULED WITH START-TIME 4 .
AT STEP 5 TASK 6 HAS BEEN SCHEDULED WITH START-TIME 4 .
AT STEP 6 TASK 2 HAS BEEN SCHEDULED WITH START-TIME 6 .
AT STEP 7 TASK 4 HAS BEEN SCHEDULED WITH START-TIME 8 .

SCHEDULE

AT STEP 1 TASK 1 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 2 TASK 7 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 3 TASK 2 HAS BEEN SCHEDULED WITH START-TIME 3 .
AT STEP 4 TASK 5 HAS BEEN SCHEDULED WITH START-TIME 4 .
AT STEP 5 TASK 3 HAS BEEN SCHEDULED WITH START-TIME 5 .
AT STEP 6 TASK 6 HAS BEEN SCHEDULED WITH START-TIME 5 .
AT STEP 7 TASK 4 HAS BEEN SCHEDULED WITH START-TIME 7 .

SCHEDULE

AT STEP 1 TASK 1 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 2 TASK 7 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 3 TASK 3 HAS BEEN SCHEDULED WITH START-TIME 3 .
AT STEP 4 TASK 5 HAS BEEN SCHEDULED WITH START-TIME 4 .
AT STEP 5 TASK 2 HAS BEEN SCHEDULED WITH START-TIME 5 .
AT STEP 6 TASK 6 HAS BEEN SCHEDULED WITH START-TIME 5 .
AT STEP 7 TASK 4 HAS BEEN SCHEDULED WITH START-TIME 7 .

SCHEDULE

AT STEP 1 TASK 1 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 2 TASK 5 HAS BEEN SCHEDULED WITH START-TIME 0 .
AT STEP 3 TASK 7 HAS BEEN SCHEDULED WITH START-TIME 1 .
AT STEP 4 TASK 2 HAS BEEN SCHEDULED WITH START-TIME 3 .
AT STEP 5 TASK 3 HAS BEEN SCHEDULED WITH START-TIME 5 .
AT STEP 6 TASK 6 HAS BEEN SCHEDULED WITH START-TIME 5 .
AT STEP 7 TASK 4 HAS BEEN SCHEDULED WITH START-TIME 7 .

```
                    SCHEDULE
AT STEP   1 TASK   1 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   2 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   3 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   1 .
AT STEP   4 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   3 .
AT STEP   5 TASK   2 HAS BEEN SCHEDULED WITH START-TIME   5 .
AT STEP   6 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   5 .
AT STEP   7 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   7 .

                    SCHEDULE
AT STEP   1 TASK   1 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   2 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   3 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   1 .
AT STEP   4 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   3 .
AT STEP   5 TASK   2 HAS BEEN SCHEDULED WITH START-TIME   4 .
AT STEP   6 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   6 .
AT STEP   7 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   8 .

                    SCHEDULE
AT STEP   1 TASK   1 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   2 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   3 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   1 .
AT STEP   4 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   3 .
AT STEP   5 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   4 .
AT STEP   6 TASK   2 HAS BEEN SCHEDULED WITH START-TIME   6 .
AT STEP   7 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   8 .

                    SCHEDULE
AT STEP   1 TASK   1 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   2 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   3 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   1 .
AT STEP   4 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   2 .
AT STEP   5 TASK   2 HAS BEEN SCHEDULED WITH START-TIME   3 .
AT STEP   6 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   5 .
AT STEP   7 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   7 .

                    SCHEDULE
AT STEP   1 TASK   1 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   2 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   3 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   1 .
AT STEP   4 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   2 .
AT STEP   5 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   3 .
AT STEP   6 TASK   2 HAS BEEN SCHEDULED WITH START-TIME   5 .
AT STEP   7 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   7 .
```

```
                    SCHEDULE
AT STEP   1 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   2 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   3 TASK   1 HAS BEEN SCHEDULED WITH START-TIME   1 .
AT STEP   4 TASK   2 HAS BEEN SCHEDULED WITH START-TIME   4 .
AT STEP   5 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   4 .
AT STEP   6 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   6 .
AT STEP   7 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   8 .

                    SCHEDULE
AT STEP   1 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   2 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   3 TASK   1 HAS BEEN SCHEDULED WITH START-TIME   1 .
AT STEP   4 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   4 .
AT STEP   5 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   4 .
AT STEP   6 TASK   2 HAS BEEN SCHEDULED WITH START-TIME   6 .
AT STEP   7 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   8 .

                    SCHEDULE
AT STEP   1 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   2 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   3 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   1 .
AT STEP   4 TASK   1 HAS BEEN SCHEDULED WITH START-TIME   2 .
AT STEP   5 TASK   2 HAS BEEN SCHEDULED WITH START-TIME   5 .
AT STEP   6 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   7 .
AT STEP   7 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   9 .

                    SCHEDULE
AT STEP   1 TASK   5 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   2 TASK   7 HAS BEEN SCHEDULED WITH START-TIME   0 .
AT STEP   3 TASK   6 HAS BEEN SCHEDULED WITH START-TIME   1 .
AT STEP   4 TASK   1 HAS BEEN SCHEDULED WITH START-TIME   2 .
AT STEP   5 TASK   3 HAS BEEN SCHEDULED WITH START-TIME   5 .
AT STEP   6 TASK   2 HAS BEEN SCHEDULED WITH START-TIME   7 .
AT STEP   7 TASK   4 HAS BEEN SCHEDULED WITH START-TIME   9 .
```

```
                    THAT-S ALL FOLKS

ENDE UROUNLO  1.40
```

Bibliographie

[1] SHRAGE L. Solving resource constrained network problems by implicit enumeration. Non preemptive case.

Opns. Res., Vol. 18, pp.263-278. (1970).

[2] JORRY A. ORDON : a scheduling algorithm for real time tasks on non preemptive resources with deadlines.

Rapport de Recherche LABORIA n° 64. (Avril 1974).

[3] JORRY A. An introduction to ORDON.

Proceedings of the 4th international Workshop on Real Time Programming, Budapest 23-25 mars 1974.

[4] JORRY A. ORDON : un algorithme pour l'ordonnancement de tâches temps-réel sur des ressources non-préemptives. Notice de programme Version PL/1 ORDON.

Rapport interne IP/216. (Avril 1974).

[5] JORRY A. OTEST : un algorithme pour la vérification des données destinées à des algorithmes d'ordonnancement. Notice de programme.

Rapport interne IP/225. (Mai 1976).

[6] JORRY A. OGEN : un algorithme pour la génération automatique de données destinées à des algorithmes d'Ordon... Notice de programme. Version 02.

Rapport interne IP/226. (Avril 1976).

[7] JORRY A. ORDON : un algorithme pour l'ordonnancement de tâches Temps-Réel sur des ressources non-préemptives. Notice de programme. Version ALGOL-60 à pile externe.

Rapport interne IP/227. (Août 1976).

ANNEXE 2

Quelques résultats récents.

TABLE DES MATIERES

A2-1) INDISPONIBILITE TEMPORAIRE DES RESSOURCES.

THEOREME 1.

Pour la prise en compte de la disparition d'une ressource déterminée, il suffit d'ajouter une tâche ayant pour date d'arrivée la date de disparition pour durée Ω, pour date critique d+Ω, avec Ω le nombre le plus grand possible. (Soit ∞ pour la planification à durée illimitée, soit D-d, avec D durée de planification , pour la planification à durée limitée), pour besoins une ressource de ce type, et de travailler ensuite comme si la ressource était toujour disponible.

Démonstration.

Si après l'ordonnancement par Ordon, la tâche fictive est placée sur la ressource correspondante, le travail est terminé. Sinon, il suffit d'intervertir le rôle joué par la ressource allouée à la tâche fictive, et par celle qui devient indisponible.

Corollaire 1.

On peut de la même façon prendre en compte la disparition de n ressources déterminées.

Démonstration.

Lors de l'ordonnancement par Ordon, une tâche fictive va se voir attribuer une ressource du type désiré pour le temps [d, d+Ω]. Si la ressource attribuée est celle qui devient indisponible à cette date, le travail est terminé. Sinon, considérons que pour toutes les ressources qui deviennent indisponibles avant la date d, la tâche fictive leur correspondant est correctement placée, la ressource allouée à la tâche fictive est donc au moins disponible sur [0,d+ε]. Il suffit donc d'intervertir le rôle de la ressource attribuée à la tâche fictive et celui de la ressource qui devient indisponible à la date d.

Par itération, on arrive ainsi à placer correctement toutes les tâches fictives.

THEOREME 2.

Pour la prise en compte de l'arrivée d'une ressource, il suffit d'ajouter une tâche ayant pour date d'arrivée 0, pour durée et date critique d'la date d'arrivée de la ressource, et pour besoin une ressource de ce type, et de travailler ensuite comme si la ressource était toujours disponible.

Démonstration.

Identique à celle du théorème 1.

Corollaire 2.

On peut ,de la même façon ,prendre en compte l'arrivée de n ressources déterminées.

Démonstration.

Identique à celle du corollaire 1, en considérant cette fois les dates d dans l'ordre décroissant.

THEOREME 3.

Si toutes les nouvelles arrivées sont postérieures aux indisponibilités précoces, on peut prendre en compte les deux phénomènes à la fois.

Démonstration.

Deux tâches fictives ne peuvent se trouver placées sur une même ressource.

Deux contraintes d'indisponibilité ne peuvent frapper la même ressource.

On peut donc procéder comme pour les corollaires 1 et 2.

THEOREME 4.

Dans le cadre du théorème 3.

La prise en compte de la disparition d'une ressource déterminée à une date d, et de l'arrivée d'une ressource équivalente déterminée à une date d+d' peut être effectuée à l'aide de l'introduction d'une seule tâche par paire. Cette tâche ayant d pour date d'arrivée, d' pour durée, d+d' pour date critique, une ressource de ce type pour besoins, on considère que ces deux ressources n'en forment qu'une seule pour l'algorithme.

Démonstration.

On procède comme pour le corollaire 1, afin de satisfaire aux contraintes d'indisponibilité anticipée.

Puis, la partie de planning suivant chaque tâche fictive est considérée comme la charge de la ressource arrivant à la date où se termine cette tâche fictive.

THEOREME 5.

S'il s'agit de recevoir, ou libérer une ressource indéterminée, l'introduction des tâches fictives selon le modèle des théorèmes 1, 2 ou 4 permet de prendre en compte toute combinaison de libérations et d'arrivées.

Démonstration.

Evidente, pour les libérations, on libère la ressource qui est allouée à cette même date à une tâche fictive ; pour les arrivées, on considère qu'il s'agit de la ressource sur laquelle une tâche fictive vient de s'achever.

A2-2) RESERVES DE SECURITE.

THEOREME 6.

Pour l'introduction de réserves de sécurité, l'introduction de k n tâches de durée unité , et celle de k tâches de durée n sont équivalentes si le principe de la constance des classes d'équivalence est vérifié . C'est-à-dire,

pour disposer de k ressources toujours disponibles, on peut dès le départ diminuer le nombre de ressources de k. On n'est donc pas obligé de considérer le cas où, à chaque instant, k ressources toujours différentes sont disponibles puisqu'il est équivalent au cas où ce sont toujours les mêmes ressources qui le sont.

Démonstration.

Considérons un planning où à chaque instant k ressources de type i sont disponibles pour servir de réserve de sécurité. L'ensemble de ces k ressources peut être variable dans le temps. Chaque ressource est à chaque instant soit disponible, soit allouée, mais ne peut être inutilisable.

Soit $r_1 \ldots r_k$ k ressources disponibles à l'instant 1.

Soit t le premier instant où l'une de ces ressources r'_t est utilisée.

Soit r''_t une ressource n'appartenant pas à $r_1 \ldots r_k$ et disponible à cet instant. r''_t existe, puisqu'il y a toujours au moins k ressources disponibles.

On peut intervertir l'usage de r'_t et r''_t à partir de l'instant t, reculant d'une unité au moins la date d'utilisation de r''_t.

On fait de même avec toutes les ressources de type r'_t augmentant ainsi d'au moins une unité la valeur de i. Ceci est toujours possible puisque, à chaque instant, au moins k ressources sont disponibles.

On obtient ainsi un planning équivalent.

En poursuivant ainsi, augmentant à chaque fois la valeur de t d'au moins une unité, on finit par obtenir un planning équivalent au premier, mais où les ressources $r_1 \ldots r_k$ sont disponibles comme réserve du début à la fin.

Il est donc possible de les réserver dès le début de l'ordonnancement, et de placer les tâches sur les ressources restantes sans plus se préoccuper des réserves.

CONCLUSION GENERALE

Il est difficile de conclure une étude qui ne l'est pas. Ce travail est la marque d'une étape, la fin des recherches effectuées dans le cadre du projet SPECTRE, nous en avons dégagé une méthode, un algorithme, une base pour un formalisme.

Si la rupture géographique est évidente, les travaux menés à Munich n'en sont pas moins la continuation : application de la méthode à un second problème, plus vaste car moins bien défini ; recherche de nouveaux algorithmes ; système d'évaluation automatique ; le tout basé sur une classification et un formalisme plus évolués.

Ce travail est donc toujours quelque chose de vivant, rappelons que chaque chapitre a son équivalent dans l'étude menée à Munich.

Alors que conclure sur des objets totalement terminés (systèmes par exemple) est relativement aisé, considérer une étude en continuel devenir est chose malaisée.

C'est pourquoi nous n'insisterons que sur la méthode développée ici, ne voulant considérer les résultats que comme une illustration, peut-être dépassée (ORDON a trois ans) et invitant le lecteur à considérer ceux nouveaux qui en découlent régulièrement ; derniers en date, ceux dus à la confrontation avec la robotisation et l'optimisation des unités de production [1] et ceux du système d'évaluation.

Nous dirons pour terminer que cette méthode de recherche de solutions étant axée sur la compréhension du problème, elle nous a permis, dans un premier temps, de structurer l'espace des solutions [2], puis de formaliser notre approche [3] et que c'est sur cette base que nous espérons pouvoir un jour passer de la description à l'explication.

REFERENCES

[1] DEPEYROT M. "Le point sur nos discussions concernant la robotique". (Juillet 1975).

[2] JORRY A. "Ordon : Un algorithme pour l'ordonnancement de tâches temps-réel sur des ressources non-préemptives. Introduction à Ordon".
Version Française : Rapport Interne LABORIA-SPECTRE IE/54. (Janvier 1975).
Version Anglaise : "International Workshop on real-time programming". Budapest. (Mars 1974). Et Rapport Interne LABORIA-SPECTRE IA/115.

[3] JORRY A. "Etude de la représentation des problèmes d'ordonnancement sous forme tensorielle. Réflexions préliminaires". Rapport Interne LABORIA-SPECTRE IE/53. (Janvier 1975).

REFERENCES BIBLIOGRAPHIQUES.

[1] MOORE J.M. "An n job, one machine sequencing algorithm for minimizing the number of late jobs". Management Science. Vol.15. N° 1. pp.102-109. (September 1968).

[2] ELMAGHRABY S.E. "The machine sequencing problem. Review and extensions" Proc. Industrial Sequencing Symposium. Naval Research Log. Quart. Vol.15. N° 2. pp.205-232. (March 1968).

[3] MARIMONT R.B. "A new method for checking the consistency of precedence matrices". Journal ACM. Vol.6. N° 2. pp.164-171. (April 1959).

[4] BAKSHI M.S.
ARORA S.R. "The sequencing problem". Management Science. Vol.16. N° 4. pp.B247-B263. (December 1969).

[5] DAY J.E.
HOTTENSTEIN M.P. "Review of sequencing research". Naval Research Log. Quart. Vol.17. N° 1. pp.11-39. (1970).

[6] SCHRAGE L. "Solving resource constrained network problems by implicit enumeration. Non preemptive case". Oper. Research. Vol.18. pp.263-278. (1970).

[7] FLORIAN M.
TREPANT P.
MAC MAHON G.B. "An implicit enumeration algorithm for the machine sequencing problem". Management Science. Vol.17. N° 12. pp.B782-B792. (August 1971).

[8] LAWLER E.L.
MOORE J.M. "A functional equation and its application to resource allocation and sequencing problem". Management Science. Vol.16. N° 1. pp.17-84. (September 1969).

[9] PRITSKER A.A.B.
WATTERS L.J.
WOLFE P.M. "Multiproject scheduling with limited resources. A 0/1 programming approach". Management Science. Vol.16. N° 1. pp.93-108. (September 1969).

[10] GALE D. "Optimal assignement in an ordered set : an application of matroid theory". Journal of Combinatorial Theory. Vol.4. pp.176-180. (1968).

Pas de [11]. (*)

[12] EPLEY D.L. "Scheduling of real time computer systems". Symposium on Computers and Automata. pp.293-303. Polytechnic Institute of Brooklyn. (April 1971).

[13] LIU C.L.
LAYLAND J.W. "Scheduling algorithms for multiprogramming in a hard real time environment". Journal ACM. Vol.20. N° 1. pp. 46-61. (January 1973).

[14] COFFMAN E.G. Jr.
GRAHAM R.L. "Optimal scheduling for two-processor systems". Acta Informatica. Vol.1. N° 1. pp.200-213. (1972).

[15] HELLER J.
LOGEMANN G. "An algorithm for the construction and evaluation of feasible schedules". Management Science. Vol.8. N° 2. pp.168-183. (January 1962).

[16] WOODGATE H.R. "Trends and developments in network planning systems". Management Informatics. Vol.2. N° 3. pp.105-119. (1973).

[17] REITER R. "Scheduling parallel computations". Journal ACM. Vol.15. N° 4. pp.590-599. (October 1968).

[18] HELLER J. "Sequencing aspects of multiprogramming". Journal ACM. Vol.8. N° 3. pp.426-439. (July 1961).

[19] SCHWARTZ E.S. "An automatic sequencing procedure with application to parallel programming". Journal ACM. Vol.8. N° 4. pp. 513-537. (1961).

(*) Les numéros correspondent à une classification interne, (numéro de cote). Certains documents communiqués personnellement ne sont pas référençables. Afin de conserver la cohérence de la numérotation, une référence vide est introduite.

[20] MARTIN D.E.
ESTRIN G.
"Experiments on models of computations and systems". IEEE. Trans. on Electr. Comp. Vol.EC 16. N° 1. pp.59-69. (February 1967).

[21] MARTIN D.E.
ESTRIN G.
"Models of computational systems. Cyclic to acyclic graph transformations". IEEE. Trans. on Electr. Comp. Vol.EC 16. N° 1. pp.70-79. (February 1967).

[22] ROUCAIROL G.
WIDORY A.
"Programmes séquentiels et parallélisme". RAIRO. N° B2. pp.5-22. (Juin 1973).

[23] REDDI S.S.
RAMAMOORTHY C.V.
"A scheduling problem". Oper. Research Quart. Vol.24. N° 3. pp.441-446. (1973).

[24] JOHNSON S.M.
"Optimal two and three stage production schedules with setup time included". Naval Research Log. Quart. Vol.1. N° 1. pp.61-68. (1954).

[25] MANNE A.S.
"On the job shop scheduling problem". Oper. Research. Vol.8. N° 2. pp.219-223. (March-April 1960).

[26] GIFFLER B.
THOMPSON G.L.
VAN NESS V.
"Numerical experience with the linear and Monte Carlo algorithms for solving production scheduling problems". Industrial Scheduling. J.F. Muth and G.L. Thompson ed. pp.21-38. Prentice Hall. (1963).

[27] FISHER H.
THOMPSON G.L.
"Probabilistic learning combinations of local job shop scheduling rules". Industrial Scheduling. J.F. Muth and G.L. Thompson ed. pp.225-251. Prentice Hall. (1963).

Pas de [28].

[29] GREENSBERG H.H.
"A branch and bound solution to the general scheduling problem". Oper. Research. Vol.16. N° 2. pp.353-361. (March-April 1968).

[30] BALAS E. "Machine sequencing via disjunctive graphs : An implici enumeration algorithm". Oper. Research. Vol.17. pp.941-957. (1969).

[31] CODD E.F. "Multiprogram scheduling. Part one and two". Comm. ACM. Vol.3. N° 6. pp.347-350. (June 1960).

[32] CODD E.F. "Multiprogram scheduling. Part three and four". Comm. ACM. Vol.3. N° 7. pp.413-418. (July 1960).

[33] DESCAMPS R. CHEVIGNON P. "Algorithmes d'optimisation pour une classe générale de problèmes d'ordonnancement avec limitations des ressources". Proc. 4th Int. Conf. on Oper. Research. pp.567-583. J. Wiley and Sons. New York. (1966).

[34] CONWAY R.W. MAXWELL W.L. OLDZIEY J.W. "Sequencing against due dates". Proc. 4th Int. Conf. on Oper. Research. pp.599-618. J. Wiley and Sons. New York. (1966).

[35] WALLER L. "A Branch and Bound and imply algorithm for an improved attack upon the job shop scheduling problem". Rapport Univ. Newcastle Upon Tyne. N° 48. (May 1973).

[36] WALLER L. "Optimal solutions to restricted n job two machine scheduling problems of minimizing makespan". Rapport Univ. Newcastle Upon Tyne. N° 51. (June 1973).

[37] GEOFFRION A.M. "An improved implicit enumeration approach for integer programming". Rapport Rand. RM 5644 PR. (June 1968). Oper. Res. Vol.17. pp.437-454. (1969).

[38] PRITSKER A.A.B. WATTERS L.J. "A 0/1 programming approach to scheduling with limited resources". Rapport Rand. RM 5561 PR. (January 1968).

[39] KAUFMAN M.T. "Anomalies in scheduling unit-time tasks". Rapport Stanford. Univ. STAN-CS-72-310. SEL 72-040. (June 1972).

[40] DREZNER S.M. VAN HORN R.L. "Design considerations for CAMCOS. A computer assisted maintenance planning and control system". Rapport Rand RM 5255 PR. (July 1967).

[41] GEOFFRION A.M. "Integer programming by implicit enumeration and Balas method". Rapport Rand. RM 4783 PR. (February 1966).

[42] GEOFFRION A.M. "Implicit enumeration using an imbedded linear program' Rapport Rand. RM 5406 PR. (September 1967).

[43] PRICE T.G. "An analysis of central processor scheduling in multiprogrammed computer systems". Rapport Stanford. Univ. STAN-CS-73-355. (October 1972).

[44] TANG D.T. WONG C.K. "A modified branch and bound strategy". Information Processing Letters. Vol.2. N° 3. pp.65-69. (August 1973)

[45] GEOFFRION A.M. "Elements of large scale mathematical programming. Part one. Concepts". Management Science. Vol.16. N° 11. pp. 652-675. (July 1970).

[46] GEOFFRION A.M. "Elements of large scale mathematical programming. Part two. Synthesis of algorithms and bibliography". Management Science. Vol.16. N° 11. pp.676-691. (July 1970).

[47] DALEY R.P. "Minimal program complexity of sequences with restricted resources". Information and Control. Vol.23. N° 4. pp. 301-312. (November 1973).

[48] SAUNDERS R.M. SCHINZINGER R. "A shrinking boundary algorithm for discrete system models". IEEE. Trans. on Syst. Sc. and Cyb. Vol.SSC 6. N° 2. pp.133-140. (April 1970).

[49] ELMAGHRABY S.E. "An algebra for the analysis of generalized activity networks". Management Science. Vol.10. N° 8. pp.494-514. (April 1964).

[50] DUDNIKOV E.E. "Detailling an optimal work schedule for an industrial complex". Automation and Remote Control. Vol.34. N° 5. Part two. pp.796-803. (1973).

[51] GONDRAN M. "Un outil pour la programmation en nombres entiers : La méthode des congruences décroissantes". RAIRO. Vol.7. N° V3. pp.35-54. (Septembre 1973).

[52] CHARLTON J.M. DEATH C.C. "A generalized machine scheduling algorithm". Oper. Research Quart. Vol.21. N° 1. pp.127-134. (1970).

[53] GARFINKEL R.S. NEMAUSER G.L. "Integer Programming". J. Wiley and Sons. (1972).

[54] VARNEY R.C. "Priority processes used for scheduling within a tree structured operating system". Information Processing Letters. pp.187-190. (1972).

[55] NEWELL G.F. "Scheduling, location, transportation, and continuum mechanics ; some simple approximations to optimization problems". SIAM. J. Appl. Math. Vol.25. N° 3. pp.346-360. (November 1973).

[56] IOFFE E.G. "Algorithm for the determination of all optimal schedules in the two operation problem of Johnson". Automation and Remote Control. Vol.34. N° 7. Part two. pp. 1123-1128. (1973).

[57] KOHLER W.H. STEIGLITZ K. "Characterization and theorical comparison of branch and bound algorithms for permutation problems". Journal ACM. Vol.21. N° 1. pp.140-156. (January 1974).

[58] SZWARC W. "Optimal elimination methods in the m × n flow shop scheduling problem". Oper. Research. Vol.21. N° 6. pp.1250-1259. (1973).

[59] BALANCHANDRAN V.
MAC GREDIE J.W.
MIKHAIL O.I. "Models of the job allocation problem in computer networks". Proc. Compcon. pp.211-214. (1973).

[60] IGNALL E.
SCHRAGE L. "Application of the branch and bound techniques to some flow shop scheduling problems". Oper. Research. Vol.13. N° 3. pp.400-413. (May 1965).

[61] SCHRAGE L. "Solving resource constrained network problems by implicit enumeration. Preemptive Case". Oper. Research. Vol. 20. N° 3. pp.668-677. (1972).

[62] LOMNICKI Z.A. "A branch and bound algorithm for the exact solution of the three machine scheduling problem". Oper. Research Quart. Vol.16. N° 1. pp.89-100. (March 1965).

[63] BELLMAN R. "Dynamic programming treatment of the travelling salesman problem". Journal ACM. Vol.9. N° 1. pp.61-63. (January 1962).

[64] GAPP W.
MANNKEKAR P.S.
MITTEN L.G. "Sequencing operations to minimize in process inventory costs". Management Science. Vol.11. N° 3. pp.476-484. (January 1965).

[65] RAMAMOORTHY C.V.
CHANDY K.M.
GONZALEZ M.J. "Optimal scheduling strategies in a multiprocessor system". IEEE. Trans. on Comp. Vol.C21. N° 2. pp.137-146. (February 1972).

[66] CROES G.A. "A method for solving travelling salesman problems". Oper. Research. Vol.6. N° 6. pp.791-812. (November-December 1958).

[67] KARG R.L.
THOMPSON G.L.
"An Heuristic approach to solving travelling salesman problems". Management and Science. Vol.10. N° 2. pp. 225-248. (January 1964).

[68] DANTZIG G.B.
FULKERSON D.R.
JOHNSON S.M.
"On a linear programming combinatorial approach to the travelling salesman problem". RAND. Corp. Santa Monica Calif. (June 1958). Oper. Res. Vol.7. N° 1. pp.58-66. (January-February 1959).

[69] HENN R.
RUB W.
"Operating strategies considering response times". International Workshop on Real Time Programming Budapest. (March 1974).

[70] SCHROTT G.
"Petri nets as a tool for comparing strategies of a real time operating system". International Workshop on Real Time Programming. Budapest. (March 1974).

[71] PIERCE J.F.
"On the truck dispatching problem". Transportation Research. Vol.3. pp.1-42. (April 1969).

[72] JONES C.H.
"An economic evaluation of job shop dispatching rules". Management Science. Vol.20. N° 3. pp.293-307. (November 1973).

[73] KARP R.M.
HELD M.
"Finite state processes and dynamic programming". SIAM. J. Appl. Math. Vol.15. N° 3. pp.693-718. (May 1967).

[74] GOLOMB S.W.
BAUMERT L.D.
"Backtrack programming". Journal ACM. Vol.12. N° 4. pp.516-524. (October 1965).

[75] CHEN K.
"A portable interpretation of duality". Hawaï International Conference on System Science. pp.658-661. (1968).

[76] PEARSON J.D.
"Duality and a decomposition technique". J. SIAM. Control. Vol.4. N° 1. pp.164-172. (1966).

[77] PEARSON J.D. "On the duality between estimation and control". J. SIAM. Control. Vol.4. N° 4. pp.594-600. (1966).

[78] ESPOSITO R. "On a relation between detection and estimation in decision theory". Information and Control. Vol.12. pp.116-120. (1968).

Pas de [79].

[80] OSAKI S. MINE H. "Linear programming considerations on Markovian decision processes with no discounting". Journal of Mathematical Analysis and Applications. Vol.26. pp.221-232. (1969).

[81] GOMORY R.E. "On the relation between integer and non integer solutions to linear programs". Proc. Nat. Academy of Sciences. Vol.53. pp.260-265. (1965).

[82] BELL E.J. "Decomposition programming : An analysis of matrix substructures". Comm. ACM. Vol.10. N° 10. pp.624-626. (October 1967).

[83] SHREIDER Y.A. "Automata and the problem of dynamic programming". Problemy Kibernetiki. Vol.5. pp.31-48. (1961).

[84] BLACKWELL D. "Discounted dynamic programming". Annals of Mathematical Statistics. Vol.36. pp.226-235. (1965).

[85] LARSON R.E. "An approach to reducing the high speed memory requirement of dynamic programming". Journal of Mathematical Analysis and Applications. Vol.11. pp.519-537. (1965).

[86] HOWARD R.A. "Dynamic programming". Management Science. Vol.12. N° 5. pp.317-348. (January 1966).

[87] DENARDO E.V. MILLER B.L. "An optimality condition for discrete dynamic programming with no discounting". The annals of Mathematical Statistics. Vol.39. N° 4. pp.1220-1227. (1968).

Pas de [88].

[89] SMALLWOOD R.D. "Optimum policy regions for Markov processes with discounting". Oper. Research. Vol.14. N° 4. pp.658-669. (1966).

[90] HOWARD R.A. "Dynamic Inference". Oper. Research. Vol.13. N° 5. pp. 712-733. (1965).

[91] FLOYD R.W. "Nondeterministic algorithms". Journal ACM. Vol.14. N° pp.636-644. (October 1967).

Pas de [92].

[93] FERNANDEZ E. BUSSEL B. "Bounds on the number of processors and time for multiprocessor optimal schedules". Rapport UCLA. (1972). μ card NTIS N° 10 P 14-109.

[94] WEST D.G. "Research on pipeline processors : A review of literature on scheduling". Rapport Lawrence Livermore. UCRL 51407. Laboratory. Univ. Calif. Liver. (June 1973).

[95] MILLER B.L. "Finite state continuous time Markov decision processes with a finite planning horizon". J. SIAM. Control. Vol. N° 2. pp.266-280. (1968).

[96] RAYMOND T.C. "Heuristic algorithm for the travelling salesman proble IBM. Journal of R. & D. Vol.13. pp.400-407. (July 1969

[97] HALL M. "Combinatorial theory". Chap.7. "Some extremal problem pp.58-65. Baisdell publishing Co. Waltham, Mass. (1967

[98] OBRUCA A.K. "Spanning tree manipulation and the travelling salesman problem". Computer Journal. Vol.10. N° 4. pp.374-377. (February 1968).

[99] NICHOLSON T.A.J. "A boundary method for planar travelling salesman problems". Oper. Research Quart. Vol.19. N° 4. pp.445- 452. (1968).

[100] WOCTTON J. "A travelling salesman algorithm". METRA. Vol.8. pp.535-542. (1969).

[101] ISAAC A.M.
TURBAN E. "Some comments on the travelling salesman problem". Oper Research. Vol.17. pp.543-546. (1969).

[102] BELLMORE M.
NEMHAUSER G.L. "The travelling salesman problem : A survey". Oper. Research. Vol.16. pp.538-558. (1968).

Pas de [103].

[104] BRENER M.A. "The formulation of some allocation and connection problems as integer programs". Naval Research Log. Quart. pp.83-95. (March 1966).

[105] MAMELAK J.S. "The placement of computer logic modules". Journal ACM. Vol.13. N° 4. pp.615-629. (October 1966).

[106] HEIDER C.H. "An N step, two variable search algorithm for the component placement problem". Naval Research Log. Quart. Vol.20. N° 4. pp.699-724. (1973).

[107] BAKER K.R.
MERTEN A.G. "Scheduling with parallel processors and linear delay costs". Naval Research Log. Quart. Vol.20. N° 4. pp. 793-804. (1973).

[108] LABETOULLE J. "Un algorithme optimal pour la gestion des processus en temps-réel". RAIRO. Vol.8. N° B1. pp.11-17. (Février 1974).

[109] SMITH G.
SEFTON I.M. "On Lion's counter example for Gotlieb's method for the construction of school time tables". Comm. ACM. Vol.17. N° 4. pp.196-202. (April 1974).

[110] KLAUS H.G. "Die Bedeutung von Konflikten für das Management von Projekten". Angewandte Informatik. N° 4. pp.171-175. (1974).

[111] CHUNG-MEI HO CHERN "A multi product joint ordering model with dependent set up costs". Management Science. Vol.20. N° 7. pp. 1081-1091. (March 1974).

[112] FRYER J.S. "Labor flexibility in multiechelon dual constraint job shops". Management Science. Vol.20. N° 7. pp.1073-1080. (March 1974).

[113] KUNREUTHER H.C.
MORTON T.E. "General planning horizons for production smoothing with deterministics demands. Part two". Management Science. Vol.20. N° 7. pp.1037-1046. (March 1974).

[114] PATTERSON J.H. "Alternate methods of project scheduling with limited resources". Naval Research Log. Quart. Vol.20. N° 4. pp.767-784. (1973).

[115] KAPUR K.C. "On max min problems". Naval Research Log. Quart. Vol. 20. N° 4. pp.639-644. (1973).

[116] FORD L.R.
FULKERSON D.R. "Flows in networks". Princeton University Press. (1962).

[117] MINIEKA E. "On computing sets of shortest paths in a graph". Comm. ACM. Vol.17. N° 6. pp.351-353. (June 1974).

[118] BELLMAN R.
KALABA R. "On k th best policies". J. Soc. Indust. Appl. Math. Vol.8. N° 4. pp.582-588. (December 1960).

[119] DIJKSTRA E.W. "A note on two problems in connexion with graphs". Numerische Mathematik. Vol.1. pp.269-271. (1959).

[120] FLOYD R.W. "Shortest path". Comm. ACM. Vol.5. N° 6. pp.345-345. (June 1962).

[121] DREYFUS S.E. "An appraisal of some shortest path algorithms". Oper. Res. Vol.17. pp.395-412. (1969).

[122] ELMAGHRABY S.E. "The theory of networks and management science. Part one". Management Science. Vol.17. N° 1. pp.1-34. (September 1970).

[123] HOFFMAN W.
PAVLEY R. "A method for the solution of the N^{th} best path problem' Journal ACM. Vol.6. N° 4. pp.506-514. (1959).

[124] CLARKE S.
KRIKORIAN A.
RAUSEN J. "Computing the N best looples paths in a network". J. Soc. Indust. Appl. Maht. Vol.11. N° 4. pp.1096-1102. (December 1963).

[125] LAROBARDIER L.M. "Planning and scheduling with DYNAMIT". Industrial Engineering. pp.38-41. (December 1973).

Pas de [126].

[127] HENN R.
LEHNHOFF S. "Strategien zur pseudo kollateralen Verarbeitung von Programmen unter Berücksichtigung vorgegebener Antwortzeiten". Rapport Techn. Univ. Munich, Fed. Rep. Germany. N° 7307. (1973).

[128] BAKER K.R.
MARTIN J.B.
"An experimental comparison of solution algorithms f the single machine tardiness problem". Naval Researc Log. Quart. Vol.21. N° 1. pp.187-200. (1974).

[129] HORN W.A.
"Some simple scheduling algorithms". Naval Research Log. Quart. Vol.21. N° 1. pp.177-186. (1974).

[130] BUTEN R.E.
SHEN V.Y.
"A scheduling model for computer systems with two cl of processors". Proc. Sagamore Computer Conference o parallel processing. pp.130-139. (1973).

[131] SCHINDLER S.
LUDTKE H.
"An approach to a restricted scheduling problem". Pr Sagamore Computer Conference on parallel processing. pp.121-129. (1973).

[132] SZWARC W.
"Mathematical aspects of the 3 * n job sequencing pr blem". Naval Research Log. Quart. Vol.21. N° 1. pp.1 153. (1974).

[133] CLARK D.
"Scheduling independent tasks on non identical paral machines to minimize mean flow time". Rapport Carneg Mellon Univ. 15213. (June 1974).

[134] BAKER K.R.
SU Z.
"Sequencing with due dates and early start times to minimize maximum tardiness". Naval Research Log. Qua Vol.21. N° 1. pp.171-176. (1974).

[135] SAVAGE S.L.
"Statistical indicators of optimality". Proc. 14th I annual symposium on switching and automata theory. p 85-91. (October 1973).

[136] WEINER P.
SAVAGE S.L.
BAGCHI A.
"Neighborhood search algorithms for finding optimal travelling salesman tours must be inefficient". Proc 5th annual ACM symposium on theory of computing. pp. 207-213. (April, May 1973).

[137] BURKOV N.
LOVETSKII S. "Methods for the solution of extremal problems of combinatorial type". Automation and Remote Control. pp.1785-1806. (1968).

[138] IVANESCU P.L. "Pseudo boolean programming and its applications". Lectures notes in Mathematics. Vol.9. Springer. (1965).

[139] SAVAGE S.L.
WEINER P.
KRONE M.J. "Towards a theory of convergent local search". Proc. 6^{t} Princeton Conference on information science and systems (1972).

[140] FRAZER W.D. "Analysis of combinatorial algorithms. A sample current algorithm". AFIPS Conference Proc. Vol.40. pp.483-491. (1972).

[141] REINGOLD E.M. "Establishing lower bounds on algorithms : A survey". AFIPS Conference Proc. Vol.40. pp.471-481. (1972).

[142] GRAHAM R.L. "Bounds on multiprocessing anomalies and related packing algorithms". AFIPS Conference Proc. Vol.40. pp.205-217. (1972).

[143] HOLLOWAY C.A.
NELSON R.T. "Job shop scheduling with due dates and variable processing time". Management Science. Vol.20. N° 9. pp. 1264-1275. (May 1974).

[144] ULLMAN J.D. "Polynomial complete scheduling problems". Rapport Berkeley Univ. N° TR 9. (March 1973).

[145] BRUNO J.
COFFMAN E.G.
SETHI R. "Scheduling independent tasks to reduce mean finishing time". Comm. ACM. Vol.17. N° 7. pp.382-387. (July 1974).

[146] HELD M.
KARP R. "A dynamic programming approach to sequencing problems". J. Soc. Indust. Appl. Math. Vol.10. N° 1. pp.196-209. (March 1962).

[147] REITER S.
SHERMAN G.
"Discrete optimizing". J. Soc. Indust. Appl. Math. Vol.13. N° 3. pp.864-889. (September 1965).

[148] FLOOD M.
"The travelling salesman problem". Oper. Research. Vol.4. N° 1. pp.61-75. (1956).

[149] LIN S.
"Computer solutions of the travelling salesman probl Bell System Tech. Journal. Vol.44. pp.2245-2269. (19

[150] PIERCE A.R.
"Bibliography on algorithms for shortest path, short spanning tree, and related circuit routing problems" ACM Sigda Newsletter. Vol.4. N° 2. pp.29-37. (June 1

[151] CHAMBERLIN D.D.
SCHLAEPPI H.P.
WLADAWSKY I.
"Experimental study of deadline scheduling for inter tive systems". IBM Journal of R. and D. pp.263-269. (May 1973).

[152] HELBIG-HANSEN K.
KRARUP J.
"Improvements of the Held-Karp algorithm for the sym tric travelling-salesman problem". Math. Programming Vol.7. pp.87-96. (1974).

[153] HELD M.
KARP R.M.
"The travelling-salesman problem and minimum spannir trees". Oper. Research. Vol.18. pp.1138-1162. (1970)

[154] PADBERG M.W.
RAO M.R.
"The travelling salesman problem and a class of poly dra of diameter two". Math. Programming. Vol.7. pp.3 45. (1974).

[155] BELLMORE M.
HONG S.
"Transformation of multisalesman problem to the star dard travelling salesman problem". Journal ACM. Vol. N° 3. pp.500-504. (July 1974).

[156] NOUR M.E.
RATHBONE D.E.
"A shortest route matrix technique". 3rd Hawaï Inter tional Conference on System Science. pp.407-410. (19

[157] KERNIGHAN B.W.
LIN S.
"An efficient heuristic procedure for partitionning graphs". Bell System Tech. Journal. pp.291-307. (February 1970).

[158] BLUM J.
"Enumeration of the square permutation in S_n". Journal of Combinatorial Theory. Vol.17. N° 7. pp.156-161. (1974).

Pas de [159].

[160] FILLMORE J.P.
WILLIAMSON S.G.
"On backtracking : A combinatorial description of the algorithm". SIAM. J. Comp. Vol.3. N° 1. pp.41-55. (March 1974).

[161] FORCINA J.L.
"Une approche simple de certains problèmes de localisation de distribution et de transport". RAIRO. N° V2. pp.39-50. (Mai 1974).

[162] KROLAK P.
FELTS W.
MARBLE G.
"A man-machine approach toward solving the travelling salesman problem". Comm. ACM. Vol.14. N° 5. pp.327-334. (May 1971).

[163] GORDON B.
MOTZKIN T.S.
"Permanents of 0,1-matrices". Journal of Combinatorial Theory. Vol.17. N° 7. pp.145-155. (1974).

[164] NEUFELD G.A.
TARTAR J.
"Graph coloring conditions for the existence of solutions to the timetable problem". Comm. ACM. Vol.17. N° 8. pp.450-453. (August 1974).

[165] JAIKUMAR R.
"An operational optimization procedure for production scheduling". Computer and Oper. Research. Vol.1. pp. 191-200. (1974).

[166] KOCZELA L.J.
"The distributed processor organization". Advances in computers. N° 9. pp.291-353. Acad. Press. (1968).

[167] RAMAMOORTHY C.V. "Optimal scheduling strategies in a multiprocessor
CHANDY K.M. system". IEEE TC. Vol.C 21. N° 2. pp.137-146.
GONZALEZ M.J. (February 1972).

Pas de [168].

[169] BORST R. "A solution technique for placing digital integrated
DEWALD C.G. circuitry on printed circuit boards". Computer and
Oper. Research. Vol.1. pp.161-180. (1974).

[S170] HABERMANN A.N. "On the harmonious co-operation of abstract machines
Thèse. Technische Hogeschool Eindhoven. (October 196

[S171] LIPTON R.J. "On synchronization primitive system". PhD. Carnegie
Mellon University. (June 1973).

[S172] BAUER H.R. "Sub problem of the maximum sequencing problem". Sta
ford University. Technical Report TR 48.
STAN-CS-72-324. SU-SEL-72-057. (1972).

[S173] HERZOG K. "Ein Programmwechsel Verfahren für cin Prozeßrechner
triebssystem". Elektron. Rechenanl. Vol.14. N° 2. pp
62-66. (1972).

[S174] LYNCH H.W. "The OS/VS2 release 2 System Resources Manager". IBM
PAGE J.B. Syst. J. Vol.14. N° 4. pp.274-291. (1974).

[S175] DEPEYROT M. "Approche barycentrique par transformation de Fourie:
d'une classe de problèmes de placement et de transpo:
C.R. Acad. Sc. Paris. Tome 279. (2 décembre 1974).
Série A. pp.823-826.

[S176] STRAUSS J.C. "An analytic model of the HASP execution task monito:
Comm. ACM. Vol.17. N° 12. pp.679-684. (December 1974

[S177] DREIER H.G.
MAASSEN F.
"Limited resources planning with a capacity adapting computer program". 4th Internet Congress. AFCET ed. Paris 30 septembre - 3 Octobre 1974. Congress Book N° 2. pp.303-307.

[S178] HAWORTH G. Mc C.
"Process scheduling by output considerations". 2nd European Seminar on Real-Time Programming. Erlangen University. (D. Fed. Rep. of Germany). (March 1972).

[S179] KAUFMAN M.T.
"An almost optimal algorithm for the assembly line scheduling problem". Stanford University. SU-SEL-73-009. μ card NTIS. AD-761-177. (January 1973).

[S180] HOLLOWAY C.A.
NELSON R.T.
"Job-shop scheduling with due dates and overtime capability". Management Science. Vol.21. N° 1. pp.68-78. (September 1974).

[S181] MANACHER G.K.
"Production and stabilization of real-time task schedules". Journal ACM. Vol.14. N° 3. pp.439-465. (July 1967)

[S182] ESKEW J.D.
PARKER R.G.
"Computational experience with a cost based algorithm for the shop scheduling problem". Oper. Research Quart. Vol.26. N° 1ii. pp.211-215. (1975).

[S183] KOCHEN M.
DEUTSCH K.W.
"A note on hierarchy and coordination an aspect of decentralization". Management Science. Vol.21. N° 1. pp. 106-114. (September 1974).

[S184] PRABHAKAR T.
"A production scheduling problem with sequencing considerations". Management Science. Vol.21. N° 1. pp.34-42. (September 1974).

[S185] NEPOMIASTCHY P.
"Application of penalty technique to solve a scheduling problem and comparison with combinatorial methods". IRIA-LABORIA 78150 Le Chesnay (FRANCE). Rapport N° 7. (January 1973).

[S186] ELMAGHRABY S.E. "The scheduling of jobs on parallel processors : A
ELSHAFEI A.N. survey". North Carolina State University.
Rapport NCSU-OR-97.
μ card NTIS AD-786-314. (Sepember 1974).

[S187] NASA "Scheduling language and algorithm development study.
Martin Marietta Final report". Parts 1. 2. 3. Appendix.
Corp. Rapports NASA-CR-140-281/282/283/363.
μ card NTIS N-74-33704/705/706/N-75-12630.

[S188] BEL G. "Optimisation dans les graphes. Flux et ordonnancement"
CAVAILLE J.B. Ecole Nationale Supérieure de l'Aéronautique et de l'Es
DELMAS J. pace. TOULOUSE. FRANCE. (1974).

[S189] CHUA Y.S. "Analysis of a feedback scheduler". SIAM. J. Comput.
BERNSTEIN A.J. Vol.3. N° 3. pp.159-176. (September 1974).

[S190] JOHNSON H.H. "Deadline scheduling for a real-time multiprocessor".
MADDISON M.S. Eurocomp. Conference Proceedings. pp.140-153. (1974).

[S191] LABETOULLE J. "Ordonnancement des processus temps-réel sur une res-
source préemptive". Thèse. Université PARIS VI (FRANCE)
(12/03/1974). IRIA-LABORIA 78150 Le Chesnay (FRANCE).
Rapport N° 74. (Mai 1974).

[S192] CHALLINE J.F. "Une approche de l'ordonnancement des échanges selon le
problème du commis voyageur". IRIA-LABORIA 78150 Le
Chesnay (FRANCE). Rapport N° 105. (Février 1975).

[S193] CONWAY R.W. "Theory of scheduling". Reading, Mass., Addison Wesley.
MAXWELL W.L. (1967).
MILLER L.W.

[S194] ASHOUR S. "Sequencing theory". Lectures notes in economics and mathematical systems. Vol.69. Springer. (1972).

[S195] COFFMAN E.G.
DENNING P.J. "Operating systems theory". Englewood Cliffs N.J., Prentice Hall. (1974).

[S196] LAPADULA L.J. "Harmonious cooperation of processes operating on a common set of data (Vol.1)". Mitre Corp. Bedford Massachusetts. Rapport MTR-2254. Vol.1. (December 1972). μ card NTIS AD-757-902.

[S197] BELL D.E. "Harmonious cooperation of processes operating on a common set of data (Vol.2)". Mitre Corp. Bedford Massachusetts. Rapport MTR-2254. Vol.2. (December 1972). μ card NTIS AD-757-903.

[S198] LAPADULA L.J. "Harmonious cooperation of processes operation on a common set of data (Vol.3)". Mitre Corp. Bedford Massachusetts. Rapport MTR-2254. Vol.3. (December 1972). μ card NTIS AD-757-904.

[S199] HENN R. "Deterministische Modelle für die Prozessorzuteilung in einer harten Realzeit-Umgebung". Thèse Techn. Univ. Munich, Fed. Rep. Germany. (22/09/1975).

[S200] DORSEY R.C.
HODGSON T.J.
RATLIFF H.D. "A production-scheduling problem with batch processing". Oper. Research. Vol.22. N° 6. pp.1271-1279. (1974).

[S201] LAWLER E.L. "On scheduling problems with deferral costs". Management Science. Vol.11. N° 2. pp.280-288. (November 1964).

[S202] O'DONOVAN T.M. "Direct solutions of M/G/1 processor sharing models". Oper. Research. Vol.22. N° 6. pp.1232-1235. (November December 1974).

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU le rapport de présentation de

. Monsieur le Professeur L. BOLLIET, U.S.M.G.

. Monsieur M. DEPEYROT, Directeur de Recherche,
ACTON, Massachussets (U.S.A.)

Monsieur Alain J O R R Y

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "GENIE INFORMATIQUE".-

Fait à Grenoble, le 27 Septembre 1976

Le Président,

Ph. TRAYNARD
Président
de l'Institut National Polytechnique