Algorithmes et ordonnancements

Lionel Dupont

▶ **To cite this version:**

Lionel Dupont. Algorithmes et ordonnancements. Modélisation et simulation. Université Joseph-Fourier - Grenoble I, 1986. Français. NNT : . tel-00322888

HAL Id: tel-00322888
https://theses.hal.science/tel-00322888

Submitted on 19 Sep 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'

Université Scientifique et Médicale de Grenoble

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE
en Mathématiques Appliquées
option: Recherche Opérationnelle

par

Lionel DUPONT

ALGORITHMES ET ORDONNANCEMENTS.

Thèse soutenue le 24 octobre 1986 devant la commission d'examen:

Président	:	**F. BRODEAU**
Examinateurs	:	**M. COSNARD**
		J. FONLUPT
		J.P. UHRY

Je remercie vivement Monsieur Jean Pierre Uhry pour la compétence avec laquelle il a dirigé ce travail.

Je remercie Monsieur François Brodeau qui me fait l'honneur de présider le jury de cette thèse.

Je suis très sensible à l'honneur que me font Messieurs Michel Cosnard et Jean Fonlupt en acceptant de participer à ce jury.

Profitant de cette occasion, je voudrai exprimer ma sympathie à tous les membres de l'équipe de Recherche Opérationnelle et à mes collègues de l'IUT.

Je remercie aussi Monsieur Iglésias et son service de reprographie pour leur collaboration dans la réalisation matérielle de ce travail.

Table des matières

Introduction

L'interêt que je porte aux problèmes d'ordonnancement qui constituent l'objet de cette thèse, date de 1980, année où, ingénieur dans une SSCI, j'ai écrit un progiciel spécifique d'ordonnancement pour une aciérie. L'ordonnancement était un ordonnancement à contraintes potentielles, plus généralement connu sous le nom de Pert dans les milieux industriels; du point de vue algorithmique, le gros des difficultées venait de la taille du projet à traiter en regard des 128k de la mémoire centrale; informatiquement, l'essentiel du travail était la gestion en conversationnel du fichier des données. Le progiciel réalisé permettait de prendre en compte 2000 tâches liées par 8000 relations de précédence, et de garder un historique du projet. Le produit achevé et livré, j'ai pensé qu'il devait être possible de reprendre algorithmes et données, afin de résoudre, sur le même type de matériel, des problèmes plus généraux et au minimum, de taille équivalente. De nombreux progiciels d'ordonnancement dès cette époque commençaient à fleurir sur le marché (Pertmaster, Planitrav...). Face à ces concurrents, le progiciel envisagé devait avoir les deux plus suivants:

- permettre de prendre en compte des horaires de travail différents selon les tâches; ce qui donne par exemple la possibilité de gérer des projets réalisés par des corps de métiers aux horaires distincts, ou bien des projets internationaux, lorsque les pays ont des législations différentes.

- permettre la prise en compte de contraintes de blocage non impératives entre les tâches; ce type de contraintes se rencontre lorsqu'on désire limiter l'intervalle de temps séparant deux tâches, sans retarder pour cela la durée du projet (afin de minimiser la durée de location de matériel par exemple).

Ces deux points restent originaux face à des progiciels micro plus récents (MacProject...).

L' étude d'un tel progiciel, restée alors sans suite, a été reprise dans le cadre de cette thèse et fait l'objet des chapitres 2 et 3. Bien que la taille mémoire des

micros-ordinateurs tende à ne plus être une contrainte pratique (un micro professionnel d'aujourd'hui ayant un minimum de 256K), la contrainte d'implantation de gros problèmes sur de petites machines a été conservée.

La seconde partie de cette thèse s'intéresse aux problèmes des représentations graphiques des résultats sous forme de diagramme à barres (ou diagramme de Gantt), et ce, de manière à minimiser le nombre de lignes du diagramme. Selon le cas, ce problème se ramène soit à la décomposition d'un ensemble muni d'une relation de préordre (problème de Dillworth), soit à la coloration d'un graphe d'intervalle. Si les algorithmes de résolution sont connus pour ces classes de graphes parfaits, nous en donnons une implémentation machine, efficace en termes de place et de temps.

Enfin, la troisième partie de cette thèse porte sur les ordonnancements avec ressources. Lorsquon a besoin de moyens matériels ou humains (les ressources) pour exécuter les tâches, et que ces moyens sont en quantité limitée, les problèmes deviennent NP-complets. Nous nous sommes restreints à la classe des problèmes où les ressources sont en quantité un et où le morcellement des tâches est interdit. Bien que ces restrictions soient fortes, ce problème demeure NP-complet, et recouvre de nombreux cas concrets, notamment certaines catégories de problèmes "d'atelier". La méthode utilisée est celle du "recuit simulé". Cette méthode, qui a fait école dans l'optimisation combinatoire de ces 3-4 dernières années, a été appliquée (notamment à Grenoble sous l'impulsion de J.P. Uhry), à divers problèmes difficiles avec une certaine efficacité. Tester le "recuit" sur micro-ordinateur, nous intéressait à double titre: d'un point de vue théorique, parce qu'il n'avait jamais été appliqué dans le domaine de l'ordonnancement; d'un point de vue pratique, pour vérifier que l'emploi de méthodes aussi "gourmandes" en temps calcul est réaliste en micro-informatique.

METHODES DU CHEMIN CRITIQUE

1. Présentation d'un exemple

Afin d'imager ce chapitre et les suivants, nous nous servirons d'un problème simplifié d'ordonnancement exposé ci-dessous.

Désireux de s'agrandir, un propriétaire décide d'adjoindre à sa maison une chambre supplémentaire avec salle d'eau. Le maître d'oeuvre du chantier, d'après son expérience, arrive à dresser le tableau des tâches requises donné ci-après.

nature des tâches	*durées*	*tâches prérequises*
1- commandes et livraisons		
C1 ciment, sable, gravier, briques	sous 8 jours	
C2 tuiles, charpente	sous 20 jours	
C3 menuiserie, vitrerie	sous 15 jours	
C4 plomberie, sanitaire, chauffage	sous 15 jours	
C5 électricité	sous 3 jours	
C6 carrelages, revêtement de sol	sous 8 jours	
2- Gros oeuvre		
G1 terrassement et fondation	30 heures	C1
G2 construction de l'ossature	40 heures + 5 jours de séchage	G1
G3 charpente	25 heures	C2, G2
G4 cloisons, plafond	30 heures	G3
G5 menuiserie, vitrerie	25 heures	C3, G4, G7
G6 pose plomberie externe, interne	30 heures	C4, G4
G7 pose tuiles	6 heures	G3
G8 pose d'un poteau électrique	1 jour	
3- Aménagements		
A1 pose électricité	16 heures	C5, G4, G8
A2 plâtre, revêtement des murs	30 heures + 2 jours de séchage	G5, A1, G6
A3 pose carrelage	6 heures + 1 jour de séchage	C6, A2
A4 pose sanitaire et radiateurs	14 heures	C4, G6, A3
4- Finitions		
F1 aménagements intérieurs	20 heures	A4
F2 aménagements extérieurs	30 heures	G6, G7

En plus, on a les contraintes temporelles suivantes. L'opération peut débuter le lundi 4 juin 1984. Une tâche nouvelle ne peut pas démarrer en cours de journée. Le maçon pressenti pour les tâches G1, G2, G4, A2 travaille 10 heures par jour du lundi au vendredi, les autres intervenants travaillent 7 heures par jour du lundi au vendredi et 4 heures le samedi. Le 11 juin (Pentecôte), le 14 juillet et le 15 août sont fériés. De plus le peintre (tâche F1) ne sera libre qu'à partir du 20 juillet, le menuisier (G5) n'est disponible que du 9 au 21 juillet. Les agents EDF viendront le 05 juillet poser le poteau (G8) et auront fini le soir même.

Sauf précision contraire, une tâche est supposée ne pas pouvoir débuter avant que toutes les tâches précédentes ne soient achevées. On dira que les liaisons sont par défaut de type Fin-Début (FD). En réalité, certaines tâches peuvent débuter avant la fin totale d'une autre (recouvrement). Dans notre exemple, on peut commencer à monter les cloisons (G4) dès que la charpente est au 3/4 posée (G3), et le plombier (G6) peut débuter dès que le maçon a fini la première cloison (G4), soit au bout de 15 h de travail. Ce type de contrainte lie les débuts des tâches et sera dite de type Début-Début (DD).

Les travaux du début de G1 à la fin de A1 exigent la location de matériel; il serait préférable de limiter le temps de location, mais sans retarder la fin du projet pour cela. De plus, pour les besoins de l'exemple, on souhaite limiter à 10 jours l'intervalle de temps séparant le début de C6 et la fin de G3.

2.Vocabulaire

PROJET: nous appellerons "projet" un ensemble d'opérations qui doivent permettre d'atteindre un objectif clairement exprimable et présentant un certain caractère d'unicité. Ainsi, construire une usine, lancer une campagne publicitaire, envoyer un homme sur la lune sont autant de projets. Par contre, fabriquer 2000 automobiles par jour relève du domaine de la fabrication en série et ne constitue pas

un projet dans notre définition.

TACHE: les opérations nécessaires pour mener à son terme le projet sont appellées tâches (ou activités). En pratique, définir une tâche n'est pas toujours immédiat: par exemple, selon la finesse du découpage, on pourra considérer la tâche "poser le toit" ou bien les tâches "poser la charpente", "poser les tuiles", "poser les gouttières"...

EVENEMENT: lorsqu'une tâche débute ou s'achève, on considère que le moment de début ou de fin est un événement. Par suite, une tâche pourra être désignée par la donnée de deux évênements.

DUREE D'EXECUTION: temps jugé nécessaire à la réalisation d'une tâche.

DUREE D'ATTENTE: laps de temps devant s'écouler entre la fin d'exécution d'une tâche et son achèvement (plâtrer une pièce demandera deux jours de travail et huit jours de séchage).

DUREE DE DECALAGE (RECOUVREMENT): durée minimale devant s'écouler entre le début de deux tâches (les électriciens pourront commencer 3 jours après les plâtriers).

CONTRAINTE POTENTIELLE: deux types de contraintes sont englobées sous ce vocable:

- les contraintes de précédence: l'exécution d'une tâche est conditionnée par la réalisation en tout ou en partie d'autres tâches. La liaison induite sera: de type FD (Fin Début) si elle lie le début d'une tâche à la fin d'une autre; de type DD (Début-Début) si elle lie les débuts des deux tâches (tâches avec recouvrement par exemple)
- les contraintes de localisation temporelle: la tâche "x" ne peut pas

commencer avant telle date et/ou doit être achevée pour telle autre date.

RESSOURCE: pour être exécutée, une tâche demande certaines ressources: de la main d'oeuvre, des machines, de l'argent. Ces ressources introduisent des contraintes disjonctives (deux tâches utilisant la même machine ne pourront s'exécuter simultanément), et des contraintes cumulatives (si l'on dispose de trois charpentiers et que cinq tâches en nécessitent un, au plus trois pourront se dérouler en même temps).

CONTRAINTE DE BLOCAGE: type de contrainte qui apparait lorsque l'on veut rendre une activité solidaire d'une activité postérieure (pas plus d'une semaine entre fin G4 et début A2).

ORDONNANCER: c'est programmer dans le temps l'exécution d'un projet, tout en respectant les contraintes et de manière à optimiser un objectif déterminé: minimiser la durée totale du projet, respecter les dates de commande, minimiser un coût etc...

ORDONNANCEMENT A CONTRAINTES POTENTIELLES: ordonnancement d'un projet dont les tâches sont soumises uniquement à des contraintes potentielles avec pour objectif de minimiser le délai total d'exécution.

CHEMIN CRITIQUE: plus long chemin joignant le début et la fin d'un projet.

MARGE TOTALE: période pendant laquelle il est possible de faire débuter une activité sans modifier la date de fin du projet.

MARGE LIBRE: période pendant laquelle il est possible de faire débuter une activité, sans que cela modifie la date de début au plus tôt des autres activités du projet.

3.Notations et définitions.

On notera G=(V,E) le graphe orienté G ayant l'ensemble de sommets V et l'ensemble d'arcs $E \in V*V$. On posera $N = |V|$ et $M = |E|$ le nombre de sommets et d'arcs de G.

On appellera graphe D-F un graphe orienté sans circuits possédant un sommet unique D sans prédécesseur (ou sommet source ou Début pour un ordonnancement) et un sommet unique F sans successeur (ou sommet puit ou Fin).

Un graphe sans circuit G=(V,E) tel que

$$\forall (x,y,z) \in V*V*V,\ xy \in E \text{ et } yz \in E \Rightarrow xz \notin E$$

sera dit minimal.

On appelle graphe transitif un graphe sans circuit G=(V,E) , tel que:

$$\forall (x,y,z) \in V*V*V,\ xy \in E,\ yz \in E \Rightarrow xz \in E.$$

Soit G=(V,E) un graphe sans circuit. On appelle numérotation isotone de G=(V,E) toute injection $f: V \rightarrow \mathbb{N}$ telle que

$$\forall (x,y) \in V*V,\ xy \in E \Rightarrow f(x) < f(y).$$

Lorsque les sommets de G sont renumérotés de 1 à N selon une fonction isotone, le nouveau graphe constitue un tri topologique de G. Quelque soit le tri topologique d'un graphe D-F, le sommet début sera numéroté 1, le sommet fin N.

4.Données historiques.

La première tentative d'ordonnancement rationnel fut celle de Gantt (cf chap3) vers 1900. On dut ensuite attendre le milieu des années 1950 pour découvrir une véritable explosion d'intérêt pour ce sujet. En Grande Bretagne, la section R.O du Central Electricity Generating Board eut à étudier les problèmes posés par la vérification d'une centrale et élabora pour ce faire une technique s'appuyant sur la recherche "d'une plus longue suite irréductible de tâches".

Un travail similaire fut entrepris aux USA par le Bureau des Projets Spéciaux de l'US Navy. Cette étude prit le nom de code "PERT" et donna lieu à un premier rapport en juillet 1958. Parallèlement l'US Air Force développa le projet PEP et la firme Du Pont de Nemours utilisa la méthode du chemin critique CPM. En France B.Roy abordait le problème un peu différemment et mettait au point la méthode des potentiels MPM. Ces travaux ont donné lieu à de nombreuses variantes. A ce jour, on les classe en deux grandes familles: la famille PERT/CPM et la famille MPM, fondées en fait toutes deux sur la recherche d'un chemin de valeur maximale dans un graphe valué. Ces méthodes et leurs dérivées ne permettent que de traiter les ordonnancements à contraintes potentielles qui demeurent les modèles de planification les plus simples. Elles restent cependant l'essentiel des méthodes appliquées dans l'industrie, bien que des modèles plus élaborés aient suscité depuis une dizaine d'années des recherches accrues. Par la suite, nous employerons le mot ordonnancement pour ordonnancement à contraintes potentielles.

La résolution d'un ordonnancement se décompose traditionnellement en 3 phases:

- définir les tâches
- créer le graphe représentatif
- calculer les durées

Nous allons les passer successivement en revue.

5. Définition des tâches.

L'approche la plus naturelle est de constituer une fiche pour chaque tâche comportant:

- son numéro ou code d'identification
- sa durée (ou ses durées)
- la liste des tâches prérequises, mentionnant recouvrements et blocages.
- la date éventuelle avant laquelle elle ne peut débuter
- la date éventuelle à laquelle elle doit être terminée.
- la date réelle de début si elle est connue
- la date réelle de fin si elle est connue

Ces deux dernières dates permettent tout à la fois d'imposer des dates strictes (cas de G8), et d'assurer un suivi de l'évolution du projet, ces dates devenant l'historique de chaque tâche.

Pour coller à la réalité, les résultats doivent tenir compte des calendriers de travail, calendriers qui peuvent différer selon le type de tâche (par exemple une tâche nécessitant 78 heures effectives correspondra à 2 semaines pour un salarié standard et à 3,25 jours pour un travail continu). C'est pourquoi on ajoutera:

- la référence au calendrier type.

Ici se pose le problème des unités de temps. Sur l'exemple, on voit qu'il y a deux mesures du temps:

1. une mesure du temps de travail proprement dit (ici l'heure)
2. une mesure des temps réels: dates (JJ/MM/AA), attente (jours). Cette mesure est celle des résultats.

On distinguera donc deux unités de temps: l'unité de travail UT pour certaines durées et l'unité calendaire UC voulue pour les résultats. Pour les calculs, les durées et dates seront ramenées à l'une de ces unités:

- durée d'exécution et de recouvrement en UT
- durée d'attente, de blocage et date en UC.

Si l'on veut un véritable outil de gestion, les résultats et dates doivent être redonnés sous la forme habituelle jour, mois, année, voire heure selon la précision de UC.

Un calendrier se définira comme une fonction de passage entre UT et UC.

6. Construction du graphe valué associé.

C'est là le point fondamental de divergence entre MPM et PERT, l'un étant construit sur les tâches, l'autre sur les étapes. Par la suite, nous utiliserons les notations ci-dessous:

- Ti représentera la tâche numéro i
- Di et Fi les étapes début et fin de Ti.
- durée(i) la durée d'exécution de Ti (processing time)
- attent(i) la durée d'attente de Ti
- dmin(i) la date minimale imposée de début (release date)
- dmax(i) la date maximale imposée de fin (due date)
- dtot(i) la date de début au plus tôt (earliest)
- dtard(i) la date de début au plus tard (latest)
- ftot(i) la date de fin au plus tôt
- ftard(i) la date de fin au plus tard
- bl(i,j) la durée de blocage entre i et j
- décal(i,j) le décalage entre le début de Ti et celui de Tj
- cal(i) le calendrier de référence de Ti

Nous considèrerons la construction du graphe par ajouts successifs en comparant les méthodes potentiels et PERT; on verra successivement:

- prise en compte des liaisons fin-début (FD),
- prise en compte des recouvrements,
- prise en compte de l'attente,
- prise en compte des contraintes de blocages,
- prise en compte des contraintes temporelles.

6.1 Liaisons Fin-Début

6.1.1 Graphe MPM

- les sommets sont constitués par les tâches Ti auxquelles on adjoint deux sommets particuliers D et F correspondant à deux tâches fictives: début et fin de projet.

- les arcs sont obtenus de la manière suivante:
  * (D,Ti) ⇔ Ti n'a pas de tâches prérequises
  * (Ti,Tj) ⇔ Ti est prérequise à Tj
  * (Ti,F) ⇔ Ti n'est prérequis à aucune tâche

- les arcs (D,Ti) ont une valuation nulle et tous les arcs (Ti,X) ont pour valeur la durée de la tâche Ti

6.1.2 Graphe PERT

- les sommets sont constitués par les étapes Di et Fi, plus deux étapes particulières D et F, début et fin du projet.

- les arcs sont de deux types:

  * les arcs réels (Di,Fi) associés aux Ti
  * les arcs fictifs que l'on associera par extension à des tâches fictives:

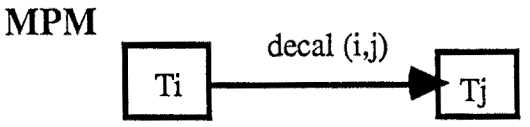
    (D,Di) ⇔ Ti n'a pas de tâches prérequises.
    (Fi,Dj) ⇔ Ti est prérequis à Tj
    (Fi,F) ⇔ Ti n'est prérequis à aucune tâche.

- les valuations sont nulles sur les arcs fictifs et les arcs réels (Di,Fi) ont pour valeur la durée de Ti.

Ce graphe double initialement les sommets du graphe MPM. Il peut toutefois être simplifié par suppression de certains arcs fictifs et par fusion de certaines étapes. Le graphe résultant est alors réputé beaucoup plus concis que le graphe MPM. Trouver le graphe comportant le nombre minimum de tâches fictives est un problème NP-complet (cf Krishnamoorthy et Deo [KR]). Par contre, il existe des algorithmes polynomiaux pour en construire avec un nombre minimal (cf Sterboule [ST]).

6.2 Recouvrement entre Ti et Tj

Dans le graphe MPM, il suffit d'affecter à l'arc (Ti, Tj) non plus la valeur durée(i) mais la valeur de décalage décal(i,j).



Dans le graphe PERT:

- soit on crée un arc fictif (Di,Dj) de valeur décal(i,j), ce qui introduit des arcs fictifs à valeur non nulle.
- soit on dédouble l'activité i en ajoutant un sommet Si entre Di et Fi puis un arc fictif (Si,Dj) avec (Di,Si) valué décal(i,j), (Si,Fi) valué durée(i)-décal(i,j).

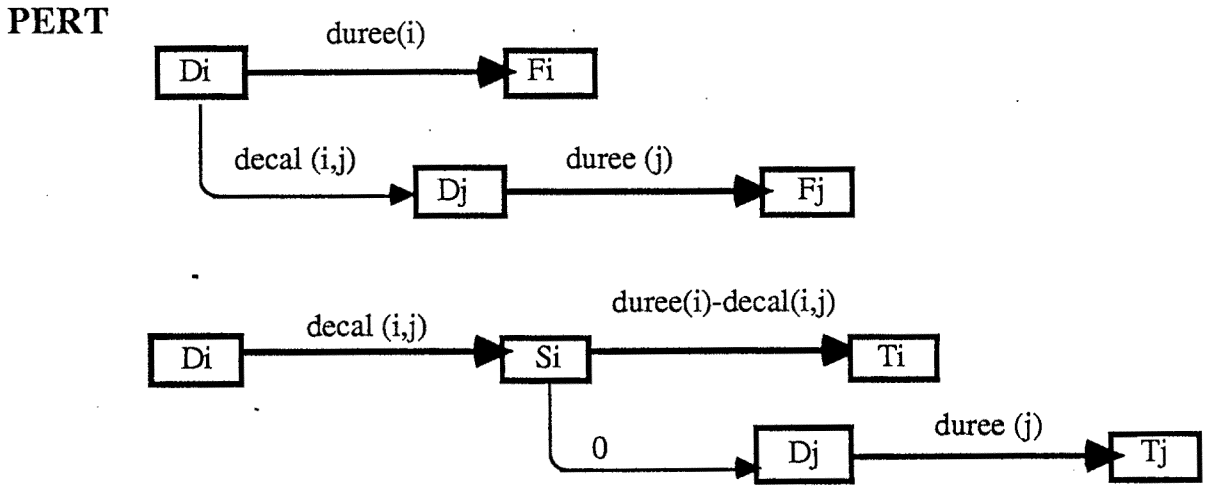


figure 1-1

NB: à ce stade de la construction, les graphes créés ne doivent pas avoir de circuit. La présence d'un circuit indiquerait une erreur de logique dans l'enchaînement des activités.

6.3 Contraintes de blocage.

Il arrive que l'on désire limiter l'intervalle de temps séparant le début (ou la fin) d'une tâche i et le début (ou fin) d'une tâche j. Par exemple, i et j nécessitent la location d'un certain matériel et l'on souhaite limiter la location à 15 jours (pas plus de 15 jours entre le début de i et la fin de j), ou bien le temps d'inutilisation (minimum de temps entre la fin de i et le début de j). Soit bl(j,i) la valeur positive de limitation ainsi introduite. Graphiquement, on est amené à construire un arc selon le tableau :

liaison		MPM		PERT	
tâche i	tâche j	arc	valeur	arc	valeur
Début	Début	Tj,Ti	-bl(j,i)	Dj,Di	-bl(j,i)
Début	Fin	Tj,Ti	-bl(j,i)-durée(j)	Fj,Di	-bl(j,i)
Fin	Début	Tj,Ti	-bl(j,i)+durée(i)	Dj,Fi	-bl(j,i)
Fin	Fin	Tj,Ti	-bl(j,i)+durée(i) -durée(j)	Fj,Fi	-bl(j,i)

Cet arc, de valeur négative est appellé aussi contrainte négative de blocage (CNB). Cette CNB sera dite inactive si elle n'a aucune répercussion sur les résultats, active dans le cas contraire. Lorsque j est un descendant de i, elle crée un circuit dans le graphe. Pour que le problème ait une solution, ce circuit doit être de longueur négative (non absorbant). Dans le cas général, ceci ne peut être garanti avant de connaître la valeur du plus long chemin de Ti à Tj. Prenons la contrainte de blocage entre G4 et A2, en supposant que l'on se fixe un maximum de 10 jours entre le début de G4 et le début de A2. Considérons les trois cas de figure suivants, où sont indiquées les dates au plus tôt.

cas 1: la CNB est inactive car dtot(A2)-dtot(G4) = 8 jours < 10 jours

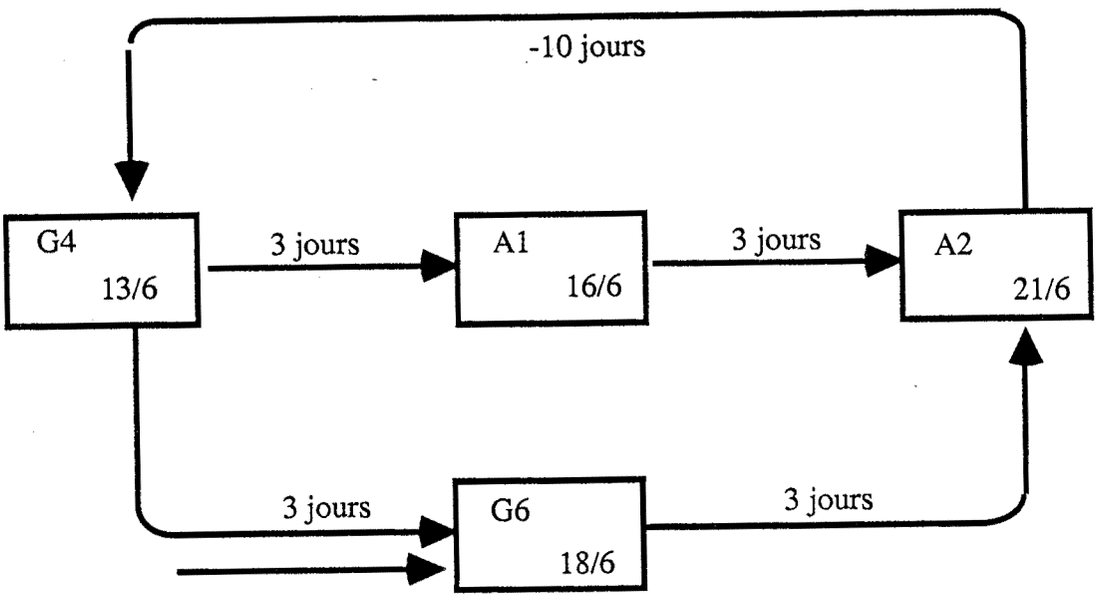


figure 1-2

cas 2: La CNB est active car dtot(A2)-dtot(G4)=16 jours et les circuits (G4,G6,A2) et (G4,A1,A2) sont négatifs. La CNB aura pour effet de bloquer le début au plus tôt de G4 au 11/6.

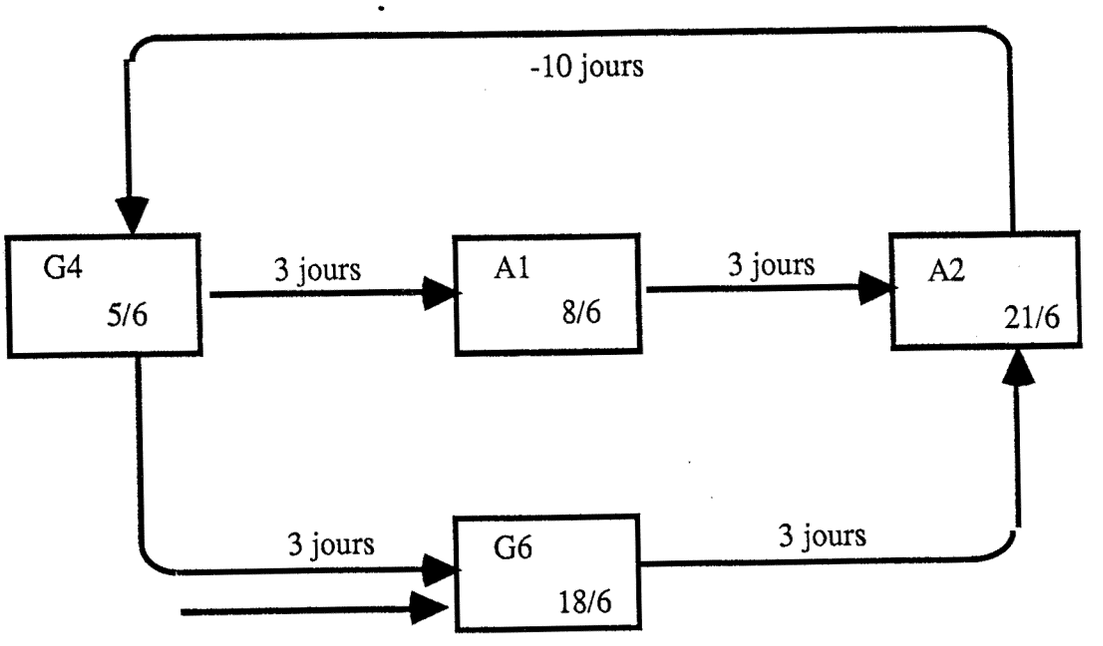


figure 1-3

cas 3: La CNB est active mais le circuit (G4,G6,A2) est positif, de valeur 1 jour et donc absorbant.

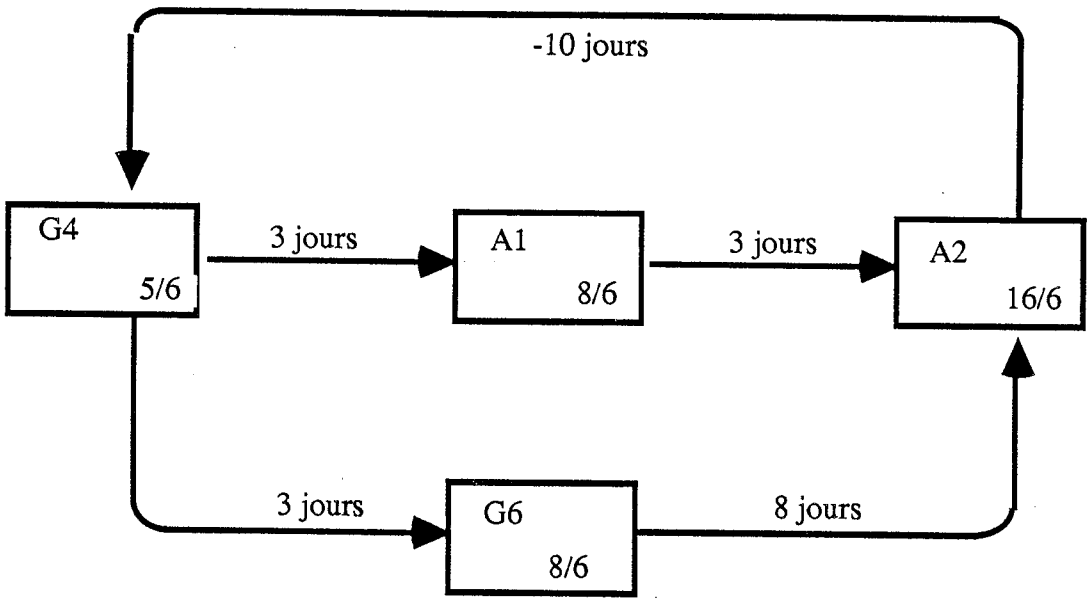


figure 1-4

Il importe de distinguer ces circuits indiquant une mauvaise estimation des durées, des circuits vu en 6.2, qui, eux indiquent une incompatibilité logique.

Selon que l'on considère que les CNB sont impératives ou non, il y a deux manières de traiter ces circuits absorbants. Si la contrainte est impérative, la valeur de l'arc (Tj,Ti) est fixée et ceci implique une réduction du chemin de Ti à Tj, donc une réévaluation des durées de certaines tâches. Ceci est du ressort du chef de projet et suspend les calculs. Dans le cas 3, cela signifie qu'il faut absolument gagner 1 jour sur le chemin G4, G6, A2. Si l'on conçoit les CNB comme un souhait (donc renégociable), ce qui est le plus fréquent, la valeur du plus long chemin détermine la valeur minimale de la CNB; il suffit de prendre cette nouvelle valeur et les calculs peuvent se poursuivre.

6.4 Contraintes optionnelles de blocage.

L'introduction d'une CNB peut avoir pour effet d'allonger la durée totale du projet. Très souvent, dans ce cas, il est préférable de relaxer la CNB et de tenir la date de fin prévue. On appellera contrainte optionnelle de blocage (COB), une CNB:

- que l'on prendra telle quelle si elle ne crée pas de circuit absorbant et si elle n'allonge pas la durée totale du projet,
- dont on acceptera d'augmenter la valeur pour qu'elle satisfasse les deux conditions précitées.

Une COB de valeur nulle équivaut à demander l'intervalle de temps minimal entre Tj et Ti.

6.5 Contraintes temporelles

La prise en compte graphique de dmin(i) peut se faire en ajoutant des arcs fictifs (D,Ti) pour MPM et (D,Di) pour PERT de valeur dmin(i); celle de dmax(i) par l'ajout d'arcs fictifs (Ti,D) de valeur durée(i)-dmax(i) ou (Fi,D) de valeur -dmax(i).

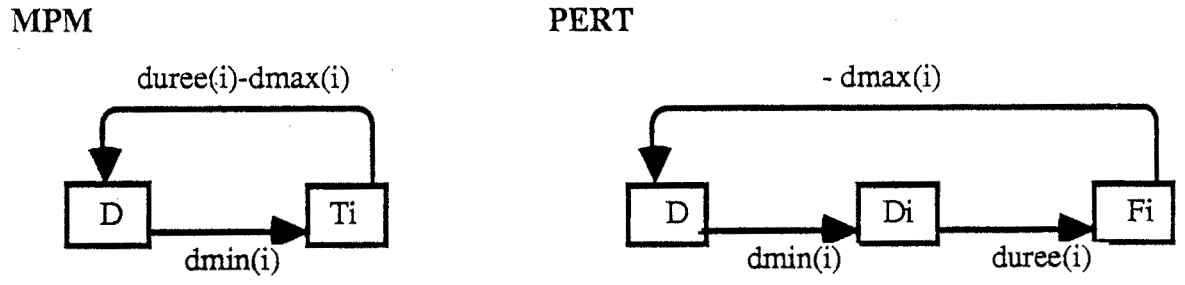


figure 1-5

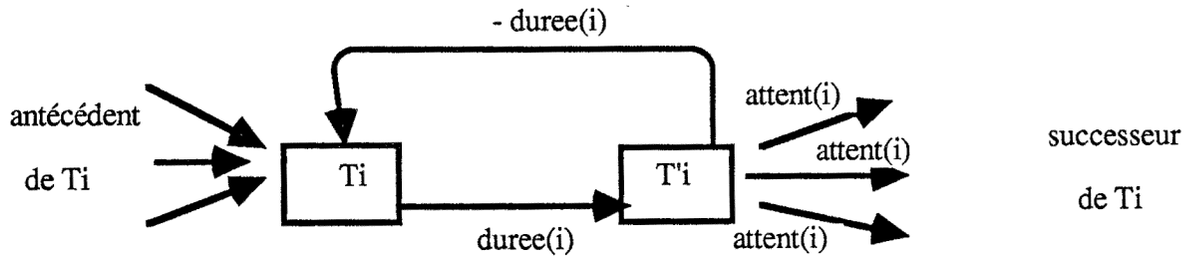
Dans les deux méthodes dmax(i) crée une CNB impérative. Il peut s'avérer après calcul, que la date au plus tôt dtot(i) soit supérieure à la date au plus tard dtard(i) impliquée par cette contrainte. Celle ci étant imposée, on gardera la valeur dtard(i) trouvée. La poursuite des calculs aura pour effet de créer un certain nombre

de tâches k telles que dtot(k) soit supérieure à dtard(k). Ces tâches, dites parfois hypercritiques, sont celles dont on doit réduire les durées afin de tenir la date de fin imposée.

6.6 L'attente.

Dans les problèmes sans notion de calendrier où durées d'exécution et durées d'attente ont la même unité, la solution la plus simple est de considérer que la durée réelle de la tâche lors des calculs, est la somme des deux. Ceci n'a plus de sens avec la prise en compte de calendriers: une activité commençant le lundi et demandant 39h de travail plus 39h de séchage sera achevée en fin de semaine alors qu'une activité de 78h (sous entendu de travail) exige une semaine de plus. Il faut donc dédoubler Ti en deux tâches distinctes, l'une représentant l'exécution du travail (de durée durée(i)), l'autre le temps d'attente (de durée attent(i)), et, pour être rigoureux, introduire une CNB créant un circuit de longueur nulle entre ces deux tâches.

MPM



PERT

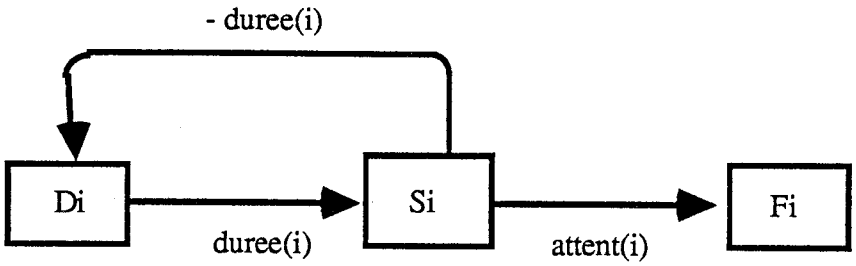


figure 1-6

7. Algorithmes fondamentaux de calculs

Une fois le graphe parfaitement défini, et ce quelle que soit la méthode, nous obtenons un graphe valué, en principe sans circuit de longueur positive (CNB), possédant deux sommets particuliers D et F. Connaissant la date de début de D, il nous faut calculer la date minimale de début de chaque Ti et de F. On reconnait là le problème de la recherche d'un chemin de valeur maximale dans un graphe valué. Ce problème est classique:

puisque le graphe est sans circuit, la méthode la plus efficace est de déterminer un tri topologique de ce graphe (ordre O(M)) et d'appliquer l'algorithme de Bellman-Ford sur le tri topologique (ordre O(M)).

Nous donnons ici les algorithmes de base permettant de résoudre un problème d'ordonnancement "d'école", sans calendrier, ni contrainte négative de blocage. Nous supposons qu'en entrée nous disposons d'un ordonnancement avec un sommet début et un sommet fin (graphe D-F) dont les sommets sont numérotés de 1 à N. On connait la date de début du projet, la durée de chaque tâche et la valuation de chaque arc.

La durée des tâches est donnée par le vecteur Durée. Pour plus de généralité et pour rendre ces algorithmes indépendants de la structure de données adoptée pour le graphe, l'accès au graphe se fera par les procédures suivantes:

INSTITUT IMAG
Informatique, Mathématiques Appliquées de Grenoble
CNRS - INPG - UJFG
MÉDIATHÈQUE
B.P. 68
38402 ST-MARTIN-D'HÈRES
FRANCE
Tél. (76) 51.46.

Liste (e: X, s: Dgm, Prédec)
où X est le numéro d'un sommet
Dgm la valeur du degré moins de X (le nombre de ses prédécesseurs)
Prédec (1..Dgm) le vecteur donnant les numéros des prédecesseurs de X

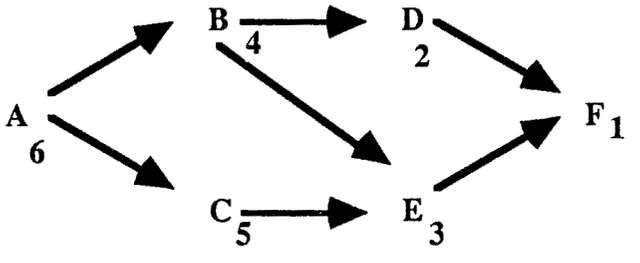
ListeValué (e: X, s: Dgm, Prédec, Valeur)
où X est le numéro d'un sommet
Dgm la valeur du degré moins de X
Prédec (1..Dgm) le vecteur donnant les numéros des prédecesseurs de X
Valeur (1..Dgm) le vecteur donnant la longueur de l' arc (Prédec(I),X)

Algorithme TriTopo

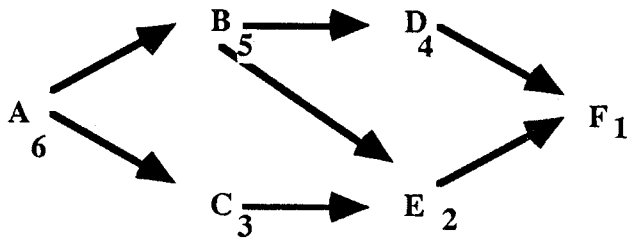
But: dans un graphe sans circuit donné par le dictionnaire des prédécesseurs de chaque sommet et dont les sommets sont numérotés de 1 à N, détermine une renumérotation de 1 à N créant un tri topologique.

Méthode:
tant qu'il existe un sommet X sans successeur, supprimer ce sommet
si le graphe restant n'est pas vide, alors il existe au moins un Circuit
sinon renuméroter les sommets dans l'ordre inverse de leur suppression.

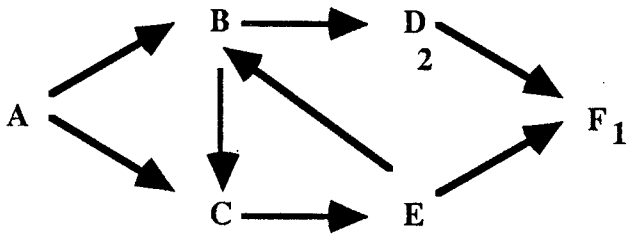
Exemple 1: le graphe est sans Circuit



mais l'on aurait aussi bien:



Exemple 2: le graphe contient un Circuit, seuls les sommets F et D peuvent être numérotés.



Implémentation

Algorithme DegrePlus (s: DPlus)

But: calcule le degré plus DPlus (X) des sommets X d'un graphe quelconque donné par le dictionnaire de ses prédécesseurs.

```
Début
   pour X de 1 à N faire DPlus(X) := 0
   pour X de 1 à N faire
     Liste (X, Dgm, Prédec)
     pour I de 1 à Dgm faire DPlus(Prédec(I)) := DPlus(Prédec(I)) + 1
   finpour
 Fin
```

Algorithme TriTopo (s: NumGraf, NumTopo, Circuit)

- NumGraf(I) donne le numéro dans le graphe de départ du sommet numéroté I dans le tri topologique .
- NumTopo(X) donne le numéro dans le tri du sommet X dans le graphe de départ
  NumGraf(I)=X ≠ NumTopo(X)=I
- Vu donne le numéro dans le tri topologique du sommet que l'on "supprime" en diminuant de 1 le degré plus de ses prédécesseurs. Si Vu=1 en fin, le graphe est sans circuit, sinon seuls les sommets donnés par NumGraf(I), I=Vu...N ont pu être numérotés.

```
Début
  DegrePlus(NumTopo)
  Suivant :=N+1;  Vu := N
  pour X de 1 à N
    si NumTopo(X)=0 alors
       Suivant := Suivant -1
       NumTopo(X):= Suivant
       NumGraf(Suivant ):=X
    finsi
  finpour
  tant que  Suivant <= Vu
    Liste(NumGraf(Vu), Dgm, Prédec)
    pour I de 1 à Dgm
      Y:= Prédec(I);   NumTopo(Y):=NumTopo(Y) -1
      si NumTopo(Y)=0 alors
        Suivant := Suivant -1
        NumTopo(Y):= Suivant
        NumGraf(Suivant ):=Y
      finsi
    finpour
    Vu := Vu-1
  fintque
  Circuit := Vu>1
Fin
```

Algorithme DateTot (e: DatDébut,NumGraf; s: DTot, FTot)

But: dans un graphe D-F dont on connait un tri topologique, connaissant la date de début du projet DatDébut, détermine les dates de début et fin au plus tôt DTot(X) et FTot(X) d'une tâche X du graphe initial pour X de 1 à N

```
Début

  pour X de 1 à N faire DTot(X) := DatDébut
  pour I de 2 à N faire
    X := NumGraf(I)
    ListeValué (X, Dgm,Prédec,Valeur)
    pour J de 1 à Dgm faire
      Y := Prédec(J);   Finy := DTot(Y) + Valeur(J)
      si DTot(X)<Finy alors DTot(X) := Finy
    finpour
  finpour
  pour X de 1 à N faire
    FTot(X) := DTot(X)
    si Durée(X) > 0 alors FTot(X) := DTot(X) + durée(X)-1
  finpour

Fin
```

Algorithme DateTard(e: DatFin, NumGraf ; s: DTard, FTard)

But: connaissant la date de fin DatFin du projet, détermine les dates de début et fin au plus tard DTard(X) et FTard(X) d'une tâche X du graphe initial pour X de 1 à N

```
Début
  pour X de 1 à N faire FTard(X) := DatFin
  pour I de N à 2 par pas -1 faire
    X := NumGraf(I)
    DTard(X) := FTard(X)
    si Durée(X)>0 alors DTard(X) := FTard(X)-Durée(X)+1
    Finy := DTard(X) -1
    ListeValué (X, Dgm, Prédec, Valeur)
    pour J de 1 à Dgm faire
      Y := Prédec(J)
      si Finy < FTard(Y) alors FTard(Y) := Finy
    finpour
  finpour
Fin
```

Algorithme Ordonnance (DatDébut)

But: calcule un problème d'ordonnancement "d'école"

```
Début
  TriTopo (V1, V2, Circuit)
  si Circuit alors écrire "Circuit détecté dans le graphe"
  sinon
    DateTot (DatDébut, V1, V2, V3)
    sauver (V2, V3)
    DatFin := V3(N)
    DateTard (DatFin, V1, V2, V3)
    sauver (V2,V3)
  finsi

Fin
```

Occupation mémoire

sous cette forme le programme nécessite:

pour définir le dictionnaire du graphe en prenant une structure classique (liste pointée...)
- 1 vecteur de dimension N pour les pointeurs
- 1 vecteur de dimension N pour les durées
- 2 vecteurs de dimension M contenant la liste des prédécesseurs et la valeur des arcs.

pour les calculs:
- 3 vecteurs de dimension N: V1, V2, V3

Si le modèle est du type FD, 1 vecteur de dimension M est suffisant et on remplacera dans les procédures Valeur(J) par Durée (Prédec(J)).

Prise en compte des calendriers.

1 Définitions.

De maniére générale, on appellera calendrier une découpe du temps selon une unité calendaire donnée ou UC (non nécessairement le jour), et date, la valeur associée à une fraction de temps d'une UC. Un calendrier de travail associe à une date le nombre d'unité de travail réalisable en cette date. Il correspond donc à une fonction à valeur positive ou nulle: soit CT(date) cette fonction. Il a pour effet de relativiser la date (début ou fin) résultant de l'ajout d'une date (début ou fin) et d'une durée.

Par exemple, pour le maçon, le calendrier de travail correspondant au mois de juin sera:

calendrier usuel						
L	M	Me	J	V	S	D
				01	02	03
04	05	06	07	08	09	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

calendrier de travail						
L	M	Me	J	V	S	D
				0	0	0
10	10	10	10	10	0	0
0	10	10	10	10	0	0
10	10	10	10	10	0	0
10	10	10	10	10	0	0

et l'on aura
début jeudi 14/6 + 20h donne fin le vend. 15/6 (inclus)
début vend. 15/6 + 20h donne fin le lundi 18/6 (inclus).
mais on aurait tout aussi bien
début jeudi 14/6 + 20h donne début autre travail le 18/6
début vend. 15/6 + 20h donne début autre travail le 19/6.

Il est possible de définir deux nouvelles fonctions calendaires à partir de CT:

- Ouvre(e: Date,Sens) qui, pour une date donnée, retourne la date du premier jour ouvré égal ou suivant cette date si Sens=1, égal ou précédant cette date si Sens=-1.

- DatCal(e: Date, Durée) qui, pour une date (début ou fin) et une durée de travail, retourne la date de fin ou début de ce travail, selon que Durée est positif ou négatif.

```
fonction Ouvre (e: Date, Sens)

 Début
      D := Date
      tant que CT(D)=0 faire
           D := D+Sens
      ftque
      Ouvre := D
 Fin
```

```
fonction DatCal (e: Date, Durée)

 Début
      si Durée=0 alors DatCal := Date
      sinon
           T := CT(Date)   D := Date
           Sens := 1
           si Durée<0 alors
                Sens := -1;  Durée := -  Durée
           finsi
           tant que T<Durée faire
                     D := D + Sens
                     T := T + CT(D)
           ftque
           DatCal := D
      finsi
 Fin
```

Ces fonctions sont définies pour un calendrier donné. Pour tenir compte du type de calendrier, on introduira un troisième parametre: Calendrier et on définira les fonctions:

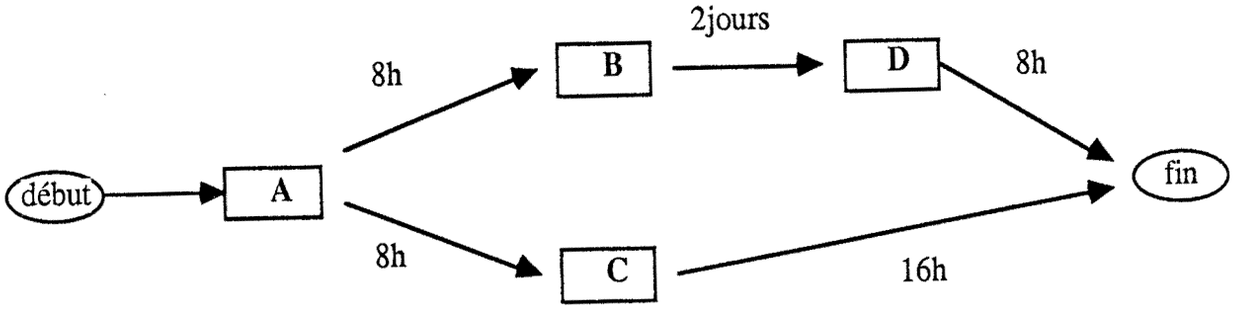
Ouvre (e: Calendrier, Date, Sens)

DatCal (e: Calendrier, Date, Durée)

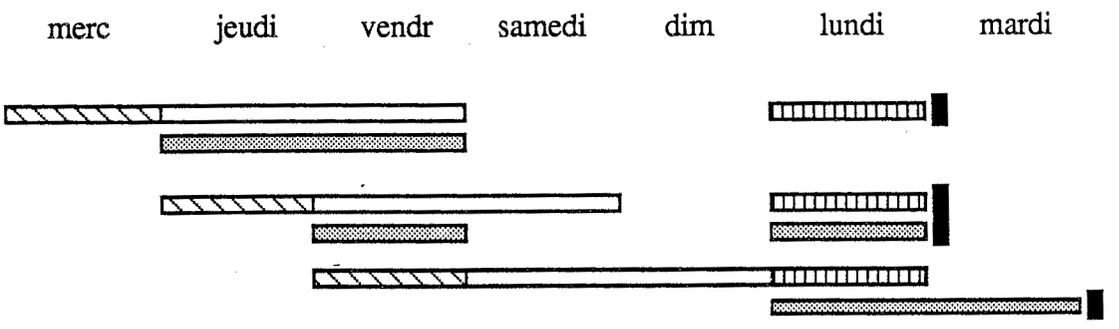
2 Particularités dues aux calendriers.

La présence de calendriers implique que le temps séparant la date de début et la date de fin d'une tâche dépend non seulement de sa durée, mais également de la date de début. Aussi on peut avoir: ftard(X)-dtard(X) ≠ ftot(X)-dtot(X) et la marge totale de X doit être calculée uniquement sur les dates au plus tôt.

Le plus long chemin de Y à X dans un graphe donné, devient lui aussi dépendant de la date de début de Y, et non plus seulement des durées. Par exemple dans le graphe suivant, on suppose que B se fait en continu et que la durée de travail est de 8h par jour du lundi au vendredi pour A, C et D.



Selon que A débute le mercredi, le jeudi ou le vendredi, on obtient les diagrammes au plus tôt suivants:



et le plus long chemin du début à la fin est de: 6 jours, 5 jours et 5 jours.

Une autre particularité due aux calendriers, réside dans la possibilité pour certaines tâches du chemin critique, d'avoir une marge de flottement non nulle. Dans l'exemple ci-dessus, si le début du projet est fixé à mercredi, on obtient:

Tâche	Date au plus tôt	Date au plus tard
A	mercredi	jeudi
B	jeudi	samedi
C	jeudi	vendredi
D	lundi	lundi

et l'on observe que seule D est une tâche critique, bien que le chemin critique soit A, B, D

3. Calcul des dates au plus tôt.

Afin de limiter le nombre d'appels aux fonctions Ouvre et DatCal (cf chap1-7), on distinguera les liaisons FD et DD. Soit l'arc (Y,X); on posera Dx, Fx la date de début et fin de X.

Soit Dx/y la date minimale à laquelle X peut débuter si l'on ne considère que Y. Si les dates Dy et Fy sont connues, on a les relations suivantes sur X:

si la liaison est FD alors
  si Durée(Y)≠0, X ne peut débuter avant la fin de Y augmentée de l'attente éventuelle et
    Dx/y := Fy + Attente(y)+1

si Durée(Y)=0, X peut débuter en même temps que Y, augmentée de l'attente éventuelle et

Dx/y := Dy + Attente(Y)

finsi

si la liaison est DD alors

si Decal(y,x) ≠ 0, X ne peut débuter avant Decal(y,x) unités de travail de Y

Dx/y := DatCal(Cal(Y), Dy, Decal(y,x))+1

si Decal(y,x) = 0, X peut débuter en même temps que Y

Dx/y := Dy

finsi

Et X ne pourra donc débuter avant le premier jour ouvré suivant le Dx/y maximum. Si de plus X a une date minimale de début imposée Dmin(X), on aura en définitif:

Dtot(X) := Ouvre [Cal(X), Max{ Dmin(X), Max (Dx/y) }, +1]

Ftot(X) := DatCal(Cal(X), Dtot(X), Durée(X))

En considérant les dates impératives, on a les deux algorithmes:

Algorithme DatTot (e:X)

But: calcule les dates au plus tôt de X connaissant celles de ses prédéccesseurs.

Début

si X a un début impératif alors

Dtot(X) := date de début

si X a une fin impérative alors Ftot(X) := date de fin

sinon Ftot(X) := DatCal(Cal(X), Dtot(X), Durée(X)) finsi

sinon

pour tout Y Pred(X), calculer Dx/y

Dtot(X) := Ouvre [Cal(X), Max{ Dmin(X), Max (Dx/y) }, +1]

Ftot(X) := DatCal (Cal(X), Dtot(X), Durée(X))

finsi

fin

Algorithme PlusTot

But: calcule l'ensemble des dates au plus tôt

Début

Dtot(0) := date début du projet

Ftot(0) := Dtot(0)

```
    pour I de 1 à N répéter
        X := NumGraf(I)
        DatTot(X)
    fin
    fin de projet := Ftot(N)
Fin
```

4.Calcul des dates au plus tard.

Soit Cy/x et Ay/x les dates maximales auxquelles Y peut commencer ou s'achever si on ne considère que X. Si on fixe maintenant Dx on a les relations:

si la liaison est FD alors
si Durée(Y)≠0, Y ne peut s'achever avant le début de X, diminué du temps d'attente éventuel; on posera:
Ay/x := Dx - Attente(Y) -1
si Durée(Y)=0, Y peut s'achever quand X débute, diminué éventuellement du temps d'attente:
Ay/x := Dx - Attente(Y)
finsi

si la liaison est DD alors
si Decal(y,x) ≠ 0, Decal(y,x) unités de travail doivent s'être réalisées avant le début de X
Cy/x := DatCal(Cal(Y), Dx-1, -Decal(y,x))
si Decal(y,x) = 0, Y peut s'achever quand X débute
Ay/x := Dx
finsi

Y ne pourra donc s'achever après le Ay/x minimum. Si de plus il y a une date maximale de fin imposée Fmax(Y), la date minimale de fin de Y donné par Ay/x est:

Fm := Min (Fmax (Y), Min(Ay/x))

Dm := DatCal(Cal(Y), Fm, -Durée(Y))

et si l'on tient compte des jours ouvrés, on obtient:

Dtard(Y) := Min (Dm, Min (Cy/x))

Ftard(Y) := DatCal (Cal(Y), Dtard(Y), Durée(Y))

En considérant les dates impératives, on a les deux algorithmes:

Algorithme DatTard (e:Y)

But: Les dates des successeurs X de Y sont connues; Dtard(Y) contient Min(Cy/x); Ftard(Y) contient Fm. On calcule les dates au plus tard de Y; puis on en répercute les conséquences sur ses prédéccesseurs Z.

Début

§ calcul des dates de Y §

si Y a un début impératif alors
Dtard(Y) := date de début
si Y a une fin impérative alors Ftard(Y) := date de fin
sinon Ftard(Y) := Ouvre(Cal(Y), Ftard(Y), -1) finsi
sinon
Ftard(Y) := Min (Fmax(Y), Ftard(Y))
Dm := DatCal(Cal(Y), Ftard(Y), -durée(Y))
si Dtard(Y)>Dm alors Dtard(Y) := Dm
Ftard(Y) := DatCal (Cal(Y), Dtard(Y), Durée(Y))
finsi

§ répercution de ces dates sur les prédécesseurs §

pour tout prédéccesseur Z de Y selon le cas:
- calculer Az/y; si Ftard(Z)>Az/y alors Ftard(Z) := Az/y
- calculer Cz/y; si Dtard(Z)>Cz/y alors Dtard(Z) := Cz/y
finpour
fin

Algorithme PlusTard

But: calcule l'ensemble des dates au plus tard

Début

Dtard(N) := date de fin du projet
Ftard(N) :=Dtard(N)
pour X de N à 1 par pas -1 répéter
DatTard(X)
finpour

Fin

Si le graphe possède M1 liaisons FD, le calcul des dates au plus tôt requiert N appels à la fonction Ouvre et N+M1 à la fonction DatCal; le calcul des dates au plus tard 2N+M1 appels à DatCal.

5. Contraintes optionnelles de blocages.

On se restreindra ici aux seules contraintes de blocage liant le début d'une tâche Y et la fin d'une tâche X. En clair, ces contraintes qui sont les plus fréquentes, expriment qu'il ne doit pas s'écouler plus de k jours entre le début de Y et la fin de X (afin de limiter l'emploi de main d'oeuvre ou le temps de location de matériel par exemple). Dans le cas où Y n'est pas un antécédent de X, on imposera que dans la numérotation topologique, le numéro de Y soit inférieur à celui de X. Ceci s'obtient facilement en introduisant un arc (Y,X) dans le graphe au moment de déterminer la numérotation, bien qu'en théorie, la contrainte de blocage crée un arc (X,Y) de valeur négative.

Les COB étant par définition optionnelles, avant de regarder dans quelle mesure elles modifient les résultats, il faut calculer les dates au plus tôt et au plus tard du projet sans COB. A ce stade, on considère que ces dates sont connues.

5.1 Prise en compte d'une contrainte optionnelle unique.

Soit une COB de k jours entre Y et X. Pour rappel, ceci équivaut à:

1/ on désire qu'il ne s'écoule pas plus de k jours entre le début de Y et la fin de X

2/ mais sans remettre en cause la date de fin du projet

3/ ni créer de circuit absorbant.

Soit Fx une date de fin possible de X ($Ftot(X) \leq Fx \leq Ftard(X)$). Le délai de k jours implique que Y ne peut débuter avant le premier jour ouvré suivant la date: Fx-k+1. Soit Dy/x cette date:

$$Dy/x := Ouvre(Cal(X), Fx-k+1, +1).$$

Si Dy/x ≤ Dtot(Y), la contrainte est sans effet.

Si Dy/x > Dtard(Y), garder Dy/x impliquerait un retard du projet. Dans ce cas, la contrainte doit être relaxée et au mieux Dy/x := Dtard(Y)

S'il existe un chemin de Y à X sur le graphe hors COB, il reste à déterminer si le circuit théorique ainsi créé est absorbant ou non. Dans le problème sans calendrier, le plus long chemin du début de Y a la fin de X est constant et indépendant des dates de Y; soit C sa longueur. Si C>k, le circuit est absorbant et C devient la valeur minimale de la COB pour que le problème soit réalisable.

Avec les calendriers, ce n'est plus le cas. La donnée importante ici, est de connaître la date maximale à laquelle Y peut débuter si Fx est fixé; autrement dit de calculer :

Dtard(Y) si Ftard(X) := Fx; soit DMy/x cette date.

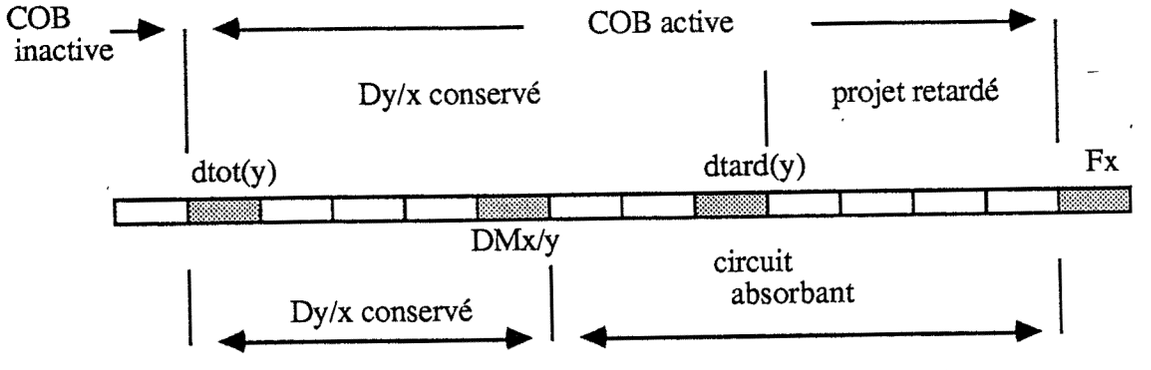
Si Dy/x>DMy/x, le circuit est absorbant. Dans ce cas, la contrainte doit être relaxée et l'on aura:

Dy/x := Min(DMy/x, Dtard(X)).

On peut remarquer, que par définition de la date au plus tard, on a toujours Dtard(Y) ≤ DMy/x. Par conséquent si Dy/x>Dtard(Y), il est inutile de calculer DMy/x.

Schéma résumé

cas1: pas de chemin de Y à X



cas 2: il existe un chemin de Y à X

Soit Dy la date retenue finalement pour Fx donné. Compte tenu que la longueur de l'intervalle (Dy,Fx) dépend dans ce cas de Fx, en faisant varier Fx entre Ftot(X) et Ftard(X), il est possible de connaître exactement les valeurs Fx pour lesquelles la COB est vérifiée, ou du moins pour lesquelles la COB est respectée au mieux. Ces valeurs peuvent former des intervalles distincts et conduiraient à des ensembles morcelés de solution. Les COB étant par définition optionnelles, on ne s'attachera pas à calculer les dates respectants au mieux ces COB, mais les dates respectant au mieux le projet hors COB et l'on déterminera les dates au plus tôt de Y respectant les dates au plus tôt de X. D'où l' algorithme:

Algorithme CobTot (e:Y,X s: Modifié)

But: détermine le début au plus tôt de Y impliquées par la COB entre Y et X.
Le logique Modifié est mis à vrai si Dtot(Y) est modifié .

```
Début
   calculer Dy/x.
   si Dtot(Y)≥ Dy/x alors la COB est inactive
   sinon
      si Dtard(Y)<Dy/x alors Dy/x := Dtard(Y)
      sinon
         s'il existe un chemin de Y à X alors
            calculer DMy/x
            si DMy/x<Dy/x alors Dy/x := DMy/x finsi
         finsi
      finsi
   finsi
   Modifié := Dtot(Y)<Dy/x
   Dtot(Y) := Dy/x
Fin
```

Considérons maintenant les implications de Y sur X. Soit Dy une date de début possible de Y (Dtot(Y)≤Dy≤Dtard(Y)). Le délai de k jours implique, que X doit être achevé au plus tard le premier jour ouvré précédant la date: Dy+k-1. Soit Fx/y cette date:

Fx/y := Ouvre(Cal(Y), Dy+k-1, -1).

Si Fx/y ≥ Ftard(X), la contrainte est sans effet.

Si Fx/y < Ftot(X), garder Fx/y impliquerait la création de tâches hypercritiques, dont il faudrait renégocier les durées. Dans ce cas, la contrainte doit être relaxée et au mieux Fy/x := Ftot(X)

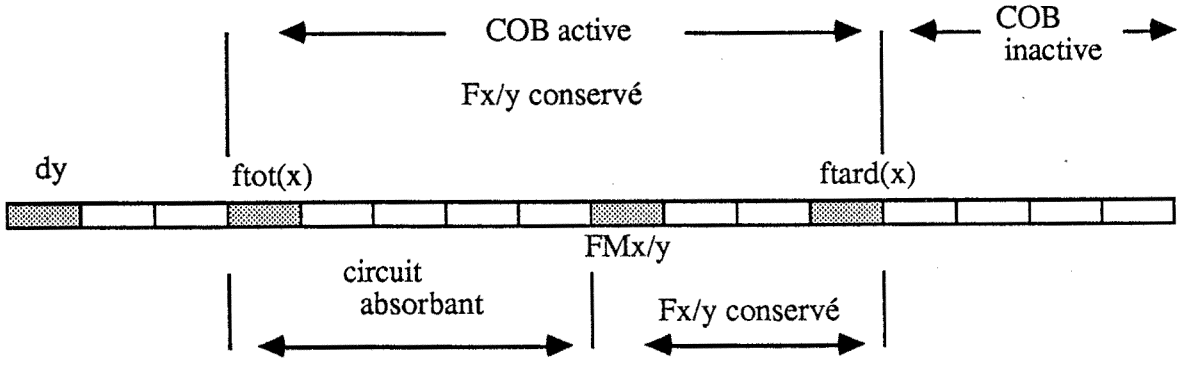
S'il existe un chemin de Y à X, la donnée importante est de connaître la date minimale à laquelle X peut finir si Dy est fixé, autrement dit de calculer Ftot(X) si Dtot(Y) := Dy; soit FMx/y cette valeur: si Fx/y<FMx/y alors le circuit est absorbant. Dans ce cas, la contrainte doit être relaxée et l'on aura: Fy/x := Max(FMy/x, Ftot(X)).

Par définition de la date au plus tôt, on a toujours Ftot(X) ≤ FMx/y. Par conséquent,

si Fx/y < Ftot(X), il est inutile de calculer FMx/y.

Schéma résumé

cas1: pas de chemin de Y à X



cas 2: il existe un chemin de Y à X

Ici, on calculera les répercutions sur les dates au plus tard de X, si l'on cherche à respecter celles de Y. D'où l'algorithme.

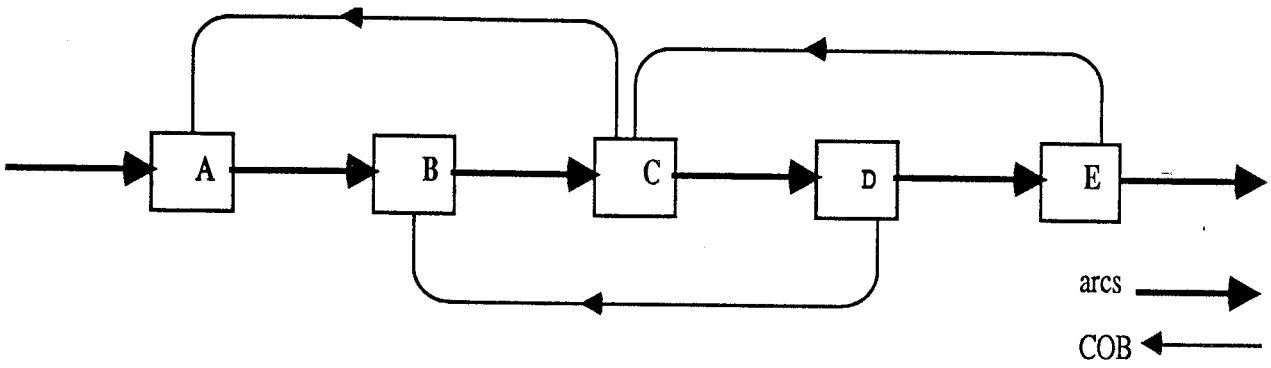
Algorithme CobTard (e: Y,X s:Modifié)

But: détermine la fin au plus tard de X impliqué par la COB entre Y et X.
Le logique Modifié est mis à vrai si Ftard(X) est modifié.

```
Début
   calculer Fy/x.
   si Ftard(X)≤ Fy/x alors la COB est inactive
   sinon
      si Ftot(Y)>Fy/x alors Fy/x := Ftot(Y)
      sinon
         s'il existe un chemin de Y à X alors
            calculer FMx/y
            si FMx/y>Fx/y alors Fx/y := FMx/y finsi
         finsi
      finsi
   finsi
   Modifié := Ftard(X)>Fx/y
   Ftard(X) := Fx/y
Fin
```

5.2 Implication globale des contraintes optionnelles.

Lorsqu'une COB modifie la date au plus tôt de Y, les dates au plus tôt des tâches de numéro (topologique) supérieure à Y peuvent être elles aussi modifiées; et à fortiori, les dates au plus tard du graphe dans son entier. Lorsque les COB sont imbriquées, comme dans le graphe ci-dessous, le recalcul des dates au plus tôt peut se répéter.La COB entre C et E peur impliquer un recalcul des dates de D, la COB B-D un recalcul de C ...



Cependant, l'algorithme reste fini. En effet, soit une COB entre Y et X. Au stade courant, les dates au plus tôt des sommets de numéro topologique inférieur à celui de X sont connues. La contrainte:

- soit laisse Dtot(Y) inchangé et l'on peut passer au sommet suivant X,
- soit augmente Dtot(Y) d'au moins une unité et l'on doit recalculer les dates au plus tôt à partir de Y. La contrainte étant optionnelle, on a Dtot(Y) ≤ Dtard(Y) et dans le pire des cas, cette borne maximale est atteinte en un nombre fini d'itérations.

Inversement, si une COB modifie la date au plus tard de X, les dates au plus tard des tâches de numéro inférieur à X peuvent être modifiées. D'où l'algorithme:

Algorithme COB

But: les dates au plus tôt et plus tard sans les COB étant connues, on prend ces dernières en compte.

Début

```
Recalcul := faux
I := 1
tant que I ≤ N faire
   X := NumGraf(I)
   Jmin := ∞
   pour tout prédécesseur Y de X
      si (Y,X) est une COB alors CobTot(Y, X, Modifié)
      si Modifié alors
          Jmin := Min (NumTopo(Y), Jmin)
          Recalcul := vrai
      finsi
   finpour
   si Recalcul alors
      si Jmin < ∞  alors I := Jmin
      DatTot(X)
   finsi
   I := I+1
fintque

si Recalcul alors
    PlusTard
finsi
Recalcul := faux
```

```
pour I de N à 1 par pas -1
   X := NumGraf(I)
   pour tout prédécesseur Y de X
      si (Y,X) est une COB alors
         CobTard(Y, X, Modifié)
         Recalcul := Recalcul ou Modifié
      finsi
   si Recalcul alors DatTard(X)
finpour
Fin
```

L'algorithme ci-dessus donne une solution, mais celle-ci peut ne pas être unique. Lorsqu'il y a interaction entre des COB, la solution trouvée peut dépendre de l'ordre dans lequel on retient les COB.

6. Dates libres.

Pour rappel, la marge libre est le retard maximum que peut prendre le début d'une tâche, sans que cela influe sur les dates au plus tôt des autres tâches du projet. On appellera "date libre", les dates servant à la détermination des marges libres. Le début libre de X est ainsi la date maximum de début de X possible lorsque les dates au plus tôt de ses successeurs sont imposées. Autrement dit:

Dlibre(X) := DatTard(X) si Dtard(Y):=Dtot(Y) pour tout Y appartenant à Succ(X)

7. calcul des augmentations de durée.

Dans les problèmes sans calendrier, les marges ont en fait deux interprétations possibles. Par exemple une marge totale de 3 jours signifie que le début de la tâche peut être reculé de 3 jours au maximum, mais tout aussi bien que la durée de cette tâche peut augmenter de 3 jours, sans que la date de fin du projet en souffre. Ici ces deux notions sont distinctes. En plus des marges données par les dates de début et fin, on calculera donc les accroissements de durée de travail correspondants. Pour cela, on définira une fonction nouvelle donnant le temps de travail total entre deux dates:

Fonction Interval (e: DatDébut, DatFin)

But: donne le temps total de travail entre les dates DatDébut et DatFin

```
Début
   trav := 0
   pour t de DatDébut à DatFin répéter trav := trav + CT(t) finpour
   Interval := trav
Fin
```

On définira ainsi:

l'accroissement libre: AccLibre := Interval (Dtot, Flibre) - Durée

l'accroissement total: AccTotal := Interval (Dtot, Ftard) - Durée.

En outre, dès qu'une tâche a lieu un jour donné, elle bloque la journée entière, même si en réalité, elle l'occupe partiellement. Il est donc parfois possible d'augmenter la durée d'une tâche sans que cela ait des répercutions sur la date au plus tôt de cette tâche. On définira de plus

l'accroissement indifférent: AccIndif := Interval (Dtot, Ftot) - Durée.

APPLICATION SUR MICRO-ORDINATEUR

Un logiciel d'ordonnancement sur micro ordinateur peut se décomposer en quatre fonctions distinctes:

1 l'introduction des données et la constitution d'un fichier de base;

2 le traitement des données relatives à la structure du graphe;

3 le traitement des données chiffrées (dates et durées) appliquées au graphe;

4 l'édition des résultats.

Constitution du fichier de base

L'utilisation d'un micro-ordinateur implique une saisie et une gestion interactive des données. Comme il est traditionnel de donner un nom codifié à chacune des tâches d'un projet, on retombe sur une gestion de fichier du type séquentiel indexé. Ce problème est du ressort de l'informatique classique et on utilisera:

- soit un gestionnaire de fichier du marché
- soit un programme spécifique .

1. Restrictions du programme proposé.

Aspect relatif au codage

Le programme doit laisser au responsable de projet la possibilité de définir un codage "parlant". Par exemple, celui-ci pourra faire correspondre aux deux premiers symboles le nom d'un responsable, aux deux suivants le type de travaux etc... Ce type de codage présente le grand intéret pratique de définir implicitement la notion de sous projet. Dans l'exemple précédent, il sera facile de fournir à chacun des

responsables les résultats des activités de son ressort et uniquement elles, ou bien de considérer ceux se rapportant à un certain type de travaux. Un code alphanumérique est donc impératif (6-8 positions semble raisonnable).

Unités de temps.

L'unité calendaire de base sera le jour exprimé sous la forme JJ/MM/AA. L'unité de travail peut être quelconque puisque la correspondance avec l'UC se fait par l'intermédiaire des calendriers. Les attentes seront en UC.

Les dates calendaires sous forme JJ/MM/AA offrent le double inconvénient d'occuper de la place mémoire et de compliquer inconsidéremment les calculs. Pour plus de commodité, on transcrira les dates calendaires sous la forme d'un entier, de telle sorte que deux entiers consécutifs correspondent à deux dates consécutives.

Les dates existantes seront transcrites de manière classique par calcul du nombre de jours écoulés depuis une date de référence arbitraire (ici 31/12/79). On pourra utiliser les algorithmes suivants, valables du 01/01/1980 au 28/2/2100. Pour information, la dernière limite est due au fait que sont bissextiles les années dont le millésime est divisible par 4, excepté les années séculaires dont le millésime n'est pas divisible par 400. Cette prise en compte alourdirait inutilement les calculs.

Fonction DatEntière (e: JOUR, MOIS, AN)

But: transforme une date calendaire de la forme JJ/MM/AA en entier

```
Début
   TRANSLAT := 29160
   MOIS := MOIS - 3
   si MOIS<0 alors
      MOIS := MOIS + 12
      A := A - 1
   finsi
   I := entier( A * 365,25 ) + entier( MOIS * 30,6 + 0,5 )
   DatEntière := I + JOUR - TRANSLAT
Fin
```

Algorithme DatCalend (e: VALEUR, s: JOUR, MOIS, AN)

But: transforme une valeur en date calendaire

```
Début
   TRANSLAT := 80
   AN := entier( (VALEUR-1)/365,25)
   VALEUR := VALEUR - entier( AN * 365,25 )
   MOIS := 1

   si VALEUR>60 alors
      VALEUR := VALEUR - 60
       MOIS := 3
   sinon
      si (A modulo 4) <> 0 alors VALEUR := VALEUR - 1
   finsi
   M1 := entier(( VALEUR - 0,5 )/30,6)
   AN := AN + TRANSLAT
   MOIS := MOIS + M1
   JOUR := VALEUR - entier( M1 * 30,6 + 0,5)
Fin
```

Exemple:

1	⇔	01/01/80					
60	⇔	29/02/80		6269	⇔	28 /02/97	
61	⇔	01/03/80	·	6270	⇔	01/03/97	

Liaisons

Sauf précision contraire, les liaisons seront considérées de type fin-début. On imposera que les liaisons Y,X de type DD (recouvrement) soient toutes données ou bien en pourcentage, ou bien en valeur de recouvrement. Ici, nous nous restreindrons aux contraintes négatives de blocage optionnelles (COB) et de type Début Y, Fin X. Elles seront exprimées en unité calendaire. En clair, il y aura au plus k jours de temps réel entre le début de la tâche Y et la fin de la tâche X. Dans la théorie, une COB entre Y et X, introduit un arc de retour (X,Y) de valeur négative.

Malgré cela, dans le fichier, cette COB sera prise en compte lors de la définition de X et on rentrera Y comme un antécédent de X, avec la valeur positive k. Ceci assure que le graphe est sans circuit (hors erreur dans les données) et que X porte un numéro supérieur à Y dans le tri topologique.

2. Contenu du fichier.

Par la suite nous considérerons que ce fichier contient pour chaque tâche:

les informations obligatoires suivantes:

- son code
- la liste des tâches immédiatement prérequises
- sa durée d'éxécution prévue
- un libellé pour les états d'édition

- lorsqu'une liaison Y,X est de type début-début (recouvrement) en parallèle à la tâche prérequise Y, le recouvrement éventuel, soit en durée, soit en pourcentage.
- lorsqu'une liaison est une COB en parallèle à Y, la valeur de blocage.

les informations facultatives suivantes:

- l'attente
- le numéro du calendrier suivi
- la date calendaire avant laquelle elle ne peut débuter
- la date calendaire à laquelle elle doit être terminée
- la date impérative à laquelle elle commence
- la date impérative à laquelle elle s'achève
- des renseignements divers.

pour pouvoir conserver un historique du projet:

- la date réelle de début
- la date réelle de fin

Un certain nombre de tests de vraisemblance doivent avoir été effectués avant le passage à la phase 2. Si le progiciel le permet, il est conseillé de faire un maximum de vérifications au cours de la phase 1 de saisie interactive:

- absence de boucles dans le graphe (interdiction d'une tâche prérequise a elle même)
- absence d'arcs multiples (une tâche ne peut être prérequise qu'une fois à une autre)
- durées positives ou nulles
- dates de début et fin cohérentes ...

Traitements relatifs au seul graphe

Ce traitement doit, outre les vérifications qui n'auraient pu être effectuées à la saisie:

- vérifier que le réseau sous jacent est complet (toute tâche indiquée dans la liste des prérequis figure dans le fichier) et qu'il existe un minimum de vraisemblance des dates entre une tâche et ses prérequises.
- le premier stade franchi, vérifier la cohérence du graphe (absence de circuis) et si la réponse est négative, guider l'opérateur pour la correction.
- une fois le graphe cohérent
  - complèter le graphe en ajoutant un sommet fin
  - déterminer une renumérotation des sommets créant un réseau topologique.

1 Création du dictionnaire des prédécesseurs

Pour gagner du temps et de la place mémoire, il est nécessaire d'éclater le fichier de base en fichiers élémentaires. Le premier fichier à constituer est le fichier des codes. Ce fichier donnera la liste des codes des tâches présentes dans le fichier de base, classés de préférence par ordre lexicographique. Cet ordre est immédiat si le fichier de base est en séquentiel indéxé, et permet de rechercher la présence d'un code donné de manière dichotomique. Par la suite, dans la représentation machine, les tâches seront données, non plus par leur code alphanumérique connu de l'utilisateur, mais par leur position dans ce premier fichier . Cette numérotation initiale des tâches de 1 à N a pour but de gagner de la place mémoire et de faciliter les manipulations du graphe en machine.

Le second fichier à constituer est le dictionnaire du graphe. Ce dictionnaire sera gardé sous la forme d'une liste pointée. Dans cette méthode, les renseignements relatifs à un sommet X sont rangés séquentiellement dans un vecteur Liste, et un second vecteur garde la position du premier renseignement (les pointeurs).

Ici, les renseignements relatifs à un sommet X ne sont pas homogènes, selon que le prédécesseur de X est une tâche avec liaison FD, DD, ou une contrainte de blocage (gardée comme arc inverse). Le vecteur Liste sera constitué de la manière suivante:

- si (Y, X) est une liaison FD, Liste(I) contient Y
- si (Y, X) est une liaison FF, Liste(I) contient -Y
  Liste(I+1) ≥ 0 contient la valeur de l'arc
- si (Y, X) est une COB, Liste(I) contient -Y
  Liste(I+1) < 0 contient la valeur de l'arc

La taille mémoire du vecteur des pointeurs est de N+1 entiers, la taille du vecteur Liste varie de M entiers si toutes les liaisons sont de type FD à 2M entiers si toutes les liaisons sont de type DD ou COB.

On introduira une tâche Début de numéro 0. Cette tâche aura une durée nulle et pour dates (début et fin, tôt et tard) la date de début du projet. Afin de réduire le graphe et les temps de calcul, il est inopportun de reprendre entièrement les tâches achevées ou ayant déja débutées. Ces tâches seront gardées dans le fichier comme des tâches ayant pour unique prédécesseur la tâche 0. Par la suite, on ne distinguera plus ces tâches des tâches ayant une date de début (ou ses dates) fixé de manière impérative.

Le dictionnaire du graphe sera créé lors d'une seule lecture du fichier de base selon l'algorithme suivant:

Algorithme Dico (sortie: Correct)

```
Début

   M := 1     Correct := juste
   Pointeur(0) := 1

   pour les tâches X de 1 à N (dans l'ordre où elles figurent dans le fichier Code) faire

      Pointeur(X) := M
      si X n'a pas de prédécesseurs ou si X a débutée (date réelle de début
          positive) alors  Liste(M) := 0
      sinon
          pour tout prédécesseur de X figurant dans le fichier
              déterminer sa position Y dans le fichier Code
              s'il n'existe pas alors Correct := faux
                 afficher le code de X et celui du prédécesseur érroné pour
                 correction manuelle ultérieure du fichier de base
              finsi
              s'il existe alors
                 si liaison FD alors  Liste(M):=Y
                 si liaison DD  alors
                    Liste(M) := -Y
                    M:=M+1  Liste(M) := valeur du recouvrement
                 finsi
```

```
                    si COB  alors
                       Liste(M) := -Y
                       M:=M+1  Liste(M) := - valeur de blocage
                    finsi

                finsi
        finsi
        M:=M+1
    fpour
    Pointeur(N+1) := M

Fin
```

A la fin de cet algorithme, le traitement peut se poursuivre si le logique Correct est vrai et M-1 donne le nombre exact d'éléments de Liste. Dans le cas contraire, la main doit être rendue à l'opérateur pour corriger le fichier de base.

Remarque: si les recouvrements sont donnés exclusivement en durée, le vecteur Liste est entièrement déterminé lors de cette lecture du fichier de base. Lorsque les recouvrements sont donnés en pourcentage, le vecteur des durées des tâches devra être connu avant de le compléter.

2 Vérification de l'absence de circuit et création du sommet Fin.

Cette partie du traitement doit vérifier l'absence de circuit dans le graphe et le compléter par l'ajout d'un sommet final, pour obtenir un graphe D-F. Ce sommet Fin devant être le seul sommet sans successeur du graphe, ce sommet sera créé en ajoutant un arc (X,Fin) à tout sommet X de degré plus nul. Ceci est aisément obtenu en modifiant l'algorithme TriTopo du chap 1-8. Avec la structure adoptée, les procédures deviennent:

```
Algorithme DegrePlus (sortie: Dplus)

   Début

      pour X de 1 à N faire Dplus(X) := 0
      Binf := Pointeur(1)  Bsup := Pointeur(N+1)

      tant que Binf<Bsup
            Y := Liste(Binf)
            si Y<0 alors Y := -Y  Binf := Binf+1 finsi
            Dplus(Y) := Dplus(Y) + 1
            Binf := Binf+1
      ftque

   Fin


Algorithme TriTopo (sortie: NumGraf, NumTopo, Vu)

   Début
      DegrePlus(NumTopo)
      Suivant:=N+1;  Vu := N
      pour X de 1 à N
         si NumTopo(X)=0 alors
               Suivant:=Suivant-1
               NumTopo(X):=Suivant
               NumGraf(Suivant):=X
               M := M+1
               Liste(M) := X
         finsi
      finpour

      N := N+1   NumTopo(N) := N
      NumGraf(N) := N
      Pointeur(N+1) := M+1    Vu := N
      tant que  Suivant ≤ Vu
         Binf := Pointeur(X)  Bsup := Pointeur(X+1)
            tant que Binf<Bsup
               Y := Liste(Binf)
               si Y<0 alors Y := -Y  Binf := Binf+1 finsi
               NumTopo(Y):=NumTopo(Y) -1
               si NumTopo(Y)=0 alors
                  Suivant:=Suivant-1
                  NumTopo(Y):=Suivant
```

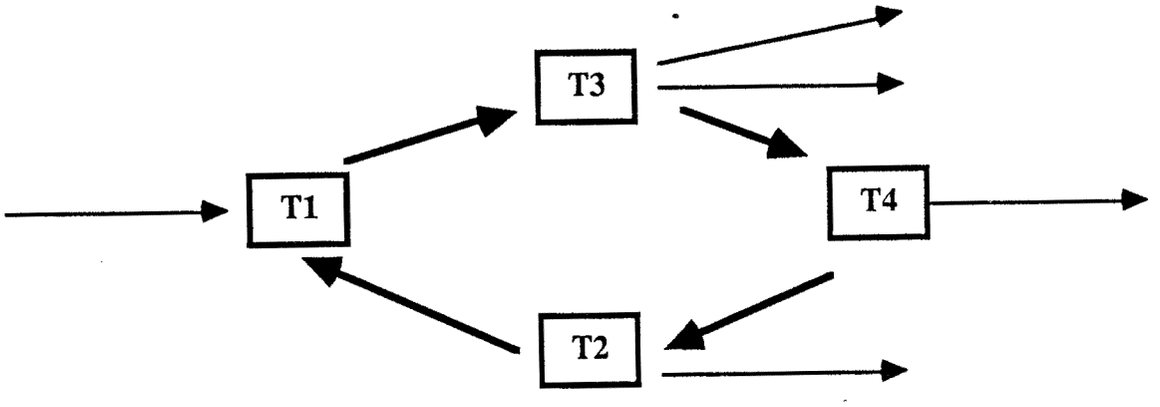
```
                NumGraf(Suivant):=Y
            finsi
            Binf := Binf+1
        ftque
    Vu := Vu-1
fintque
```

Fin

A la fin de cet algorithme, si Vu<2 alors le graphe est sans circuit. Le graphe complet contient N sommets, et Liste, M éléments.Dans le cas contraire, le fichier doit être corrigé.

3. Correction du graphe (suppression des circuits)

L'existence de circuits dans un graphe d'ordonnancement résulte de la présence d'arcs erronés. La détermination de ces arcs, sauf cas d'espèce, ne peut se faire automatiquement et requiert la présence d'un opérateur humain. Par exemple, dans le schéma suivant



chacun des arcs (T1,T3), (T3,T4), (T4,T2), (T2,T1) supprime le circuit et seule la signification logique des arcs peut déterminer lequel supprimer. Pour guider l'opérateur dans son travail, on peut envisager deux moyens. Soit lui fournir une liste

d'arcs à vérifier, soit lui fournir une liste de circuits.

3.1 liste d'arcs

Tout arc appartenant à un circuit est, dans l'absolu, susceptible d'être erroné. Un tel arc sera dit listable. Trouver l'ensemble des arcs listables est un problème facile. On vérifiera par exemple, si après suppression de l'arc (x,y), il reste possible par marquage arrière d'atteindre y à partir de x.

Du point de vue pratique, cependant, cette méthode peut s'avérer totalement inefficace. En effet, un arc erroné peut engendrer plusieurs circuits et par suite de nombreux arcs listables (tous les arcs dans le pire des cas).

On peut imaginer des heuristiques pour donner à l'opérateur en premier lieu les arcs ayant les plus "fortes chances" d'être faux. Par exemple, on observe que si l'arc erroné est unique, sa suppression implique nécessairement celle de tous les arcs listables du graphe. L'heuristique de choix serait alors:

- calculer le nombre C(i) d'arcs listables disparus avec la suppression de l'arc listable i
- afficher les arcs par C(i) strictement positif décroissant,

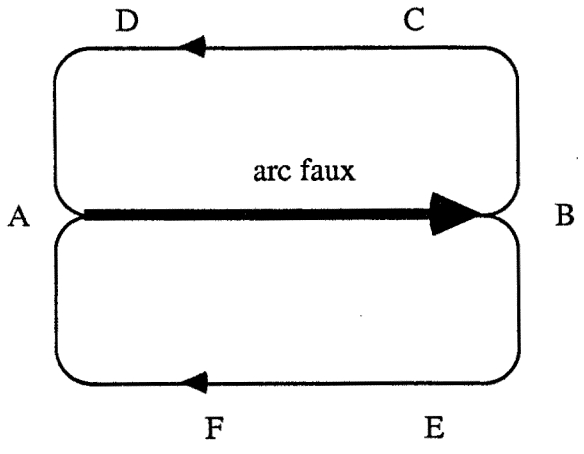
l'opérateur ayant le choix d'arrêter les vérifications dès qu'il a trouvé au moins un arc erroné, pour correction du fichier de base et relance du traitement.

Outre les temps de calcul de ces algorithmes, le problème de toute heuristique demeure: on ne peut garantir l'efficacité de la méthode de manière générale. C'est pourquoi, on ne développera pas plus cette approche.

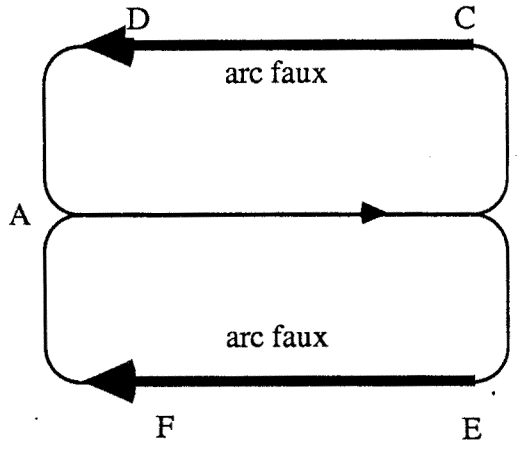
3.2 liste de circuits

Donner une liste de circuits est une méthode plus féconde. Et vis à vis de l'opérateur, elle a le petit plus psychologique de lui montrer de visu le circuit. On peut

envisager de donner un circuit à la fois. Comme chaque circuit contient nécessairement au moins un arc faux, il faudra au plus autant de cycles détection, correction du fichier de base, relance du programme que d'arcs erronés. Pour accélérer le processus, sans pour cela surcharger l'opérateur, la solution est de fournir une liste de circuits, telle que tout circuit présente au moins un arc erroné nouveau. L'optimum serait qu'à l'inverse tout arc erroné figure dans cette liste. Si l'on considère les deux schémas suivants, on se rend compte que ce problème est ambigu.



Cas 1



Cas 2

cas 1 la liste ne peut contenir qu'un des deux circuits

cas 2 la liste doit contenir les deux

et le choix dépend de la position de(s) arc(s) erroné(s) que nous ignorons par hypothèse. En fait, au mieux il n'est possible que de fournir une liste maximale de circuits distincts par les arcs.

Algorithme Circuits

Méthode (basée sur une exploration en profondeur):
on cherche à créer un chemin remontant à partir d'une racine

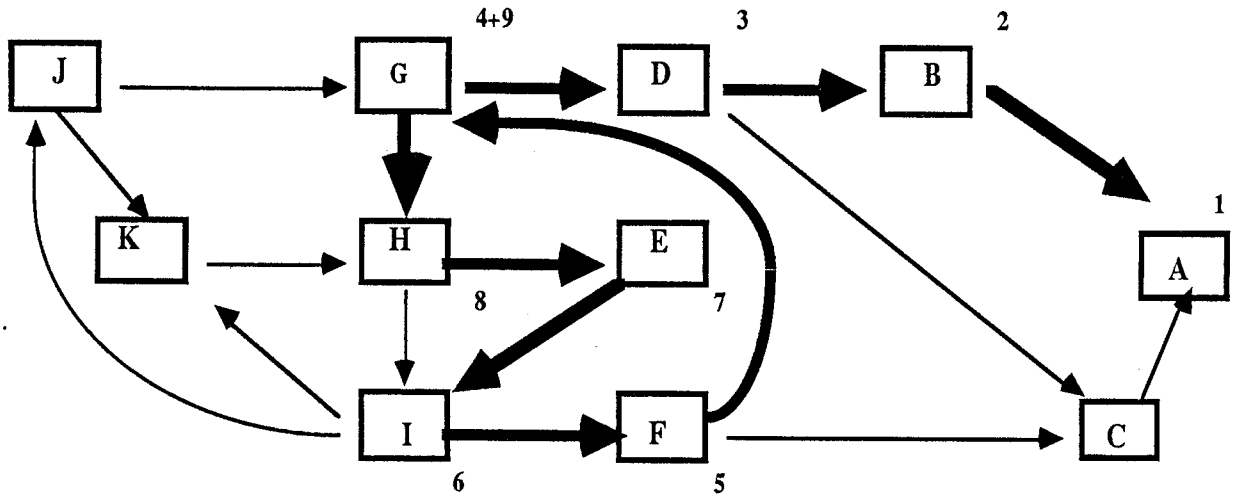
à tout sommet X on associe une marque MR
MR(X) = -1 si l'exploration de X est terminée
MR(X) = 0 si elle n'est pas débutée (marque initiale)
MR(X) > m si X est le sommet en m(x)-ième position dans l'exploration en cours
initialement les sommets ayant pu être numérotés par TriTopo sont marqués explorés

Début

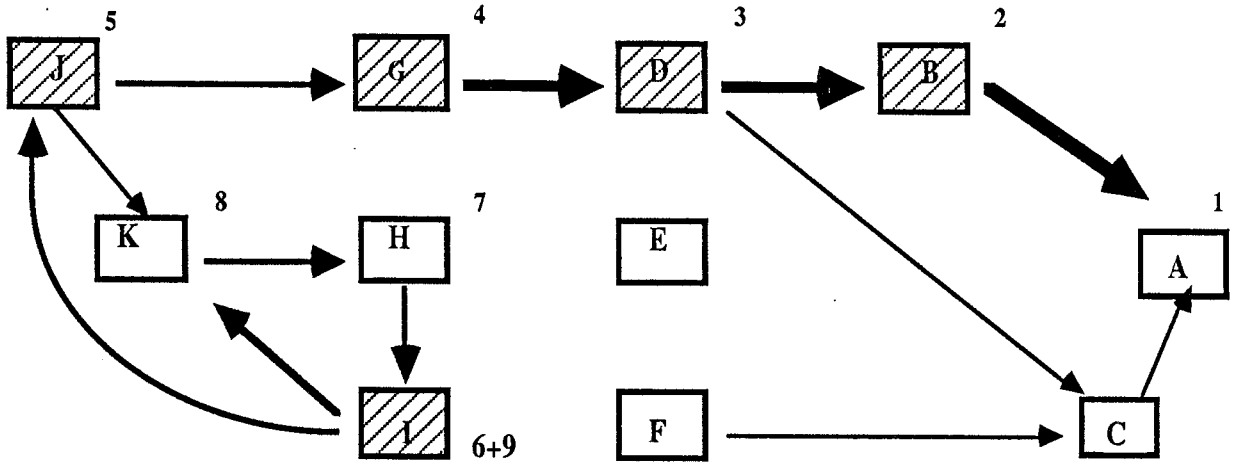
tant qu'il existe un sommet X non fini d'explorer faire
prendre ce sommet pour racine (chemin réduit à X), MR(X):=1
tant que le chemin est non vide faire
soit Y le sommet début du chemin
si Y est une source, Y est fini d'explorer: le marquer -1 et
le retirer du graphe
sinon
soit Z un ou le prédécesseur de Y de marque maximum. on a les 3 cas:
1/ MR(Z)=-1 et tous les prédécesseurs de Y ont été explorés.
On abandonne l'exploration sur Y;
marquer -1 à Y et le retirer du chemin.
2/ MR(Z)=0 on poursuit l'exploration sur Z. Ajouter Z au chemin
MR(Z) := MR(Y)+1 (Z devient le nouveau début)
3/ MR(Z)>0, Z forme un circuit avec ses ascendants sur le chemin
(les sommets S tels que MR(S)>MR(Z))
Effacer les arcs du circuit. Retirer les sommets S du chemin
(MR(S) := 0) Reprendre l'exploration à partir de Z
finsi
fintq
fintq

Fin

exemple: en choisissant systématiquement les sommets par ordre alphabétique, on obtient:



pas1: en partant de A, on marque successivement les sommets B, D, G,F, I, E et H qui marque G déjà atteint. On supprime les arcs du circuit et l'on repart de G.



pas 2: on marque J, I, H, K qui marque I déjà atteint. On supprime le circuit et on repart de I.
pas 3: on abandonne successivement I, J, G, D, B et on repart de A.
pas 4: on marque C, F et l'on abandonne F, C, A
pas 5: on abandonne E.
pas 6: on marque H, K et on abandonne K,H.

implantation

Pour augmenter la lisibilité de l'gorithme Circuits, on ajoutera deux procédures de gestion de la liste.

Algorithme ListPred (e:X, s: Dgm, Pred)

But: donne la liste des prédécesseurs du sommet X, que la liaison soit FD, DD ou COB; le sommet 0 est ignoré.

```
Début
   Binf := Pointeur(X)   Bsup := Pointeur(X+1)
   I := Binf    Dgm  := 0
   tant que I<Bsup faire
      Dgm  := Dgm + 1
      S := liste(I)
      si S<0 alors
         Pred(Dgm) := -S
         I := I+2
      finsi
      si S>0 alors
         Pred(Dgm) := S
         I :=I+1
      finsi
   ftque
Fin
```

Algorithme Suppress (e: Y,X)

But: supprime l'arc (Y,X) du graphe en le remplaçant par l'arc (0,X)

```
   début

      Binf := Pointeur(X)
      Bsup := Pointeur(X+1)
      tant que Binf < Bsup faire
         Z := Liste(Binf)
         I := 0
```

```
        si Z<0 alors
           Z:=-Z  I := 1
        finsi


        si Z=Y alors
           Liste(Binf) := 0
           Liste(Binf+I) := 0
           Binf := Bsup
        finsi
        Binf := Binf+I+1
     fintq

  fin
```

Algorithme Circuits (e: Vu, Chemin)

où Vu et Chemin reprennent les sorties Vu et NumGraf de TriTopo, donnant les sommets ayant pu être renumérotés.

```
Début

 § initialisation de MR - les sommets numérotés par TriTopo sont explorés §

   pour X de 1 à N faire MR(X) := 0
   pour I de Vu à N faire
      X := Chemin(I)
      MR(X):= -1
   finpour

§  recherche d'un sommet non exploré et exploration  §

   pour X de 1 à NBSOM faire
      tant que MR(X)=0
         MR(X) := 1
         Traité := 1
         Chemin(Traité) := X

         tant que Traité>0
            Y := Chemin(Traité)
            ListePred(Y, NB, P)
```

```
            si NB=0 alors  Cas  := -1
            sinon
               Z := P(1)
               pour I de 2 à NB répéter
                        si MR(Z)<MR(P(I)) alors Z := P(I)
               finpour
                Cas  := MR(Z)
            finsi

             si  Cas =-1 alors
               MR(Y) := -1
               Traité := Traité-1
            finsi


            si  Cas >0 alors
               Chemin(Traité+1) := Z
               écrire 'circuit détecté : '
               pour I de  Cas  à Traité répéter
                        écrire CODE(Chemin(I))
                        Suppress(Chemin(I), Chemin(I+1))
               finpour
               pour I de  Cas +1 à Traité répéter
                        MR(I) := 0
               finpour
               Traité :=  Cas
            finsi
            si Cas = 0 alors
               Traité := Traité+1
               MR(Z) := Traité
               Chemin(Traité) := Z
            finsi
         fintq
   fintq
 finpour

Fin
```

4. Occupation mémoire et sauvegarde des fichiers.

L'occupation mémoire du traitement relatif au seul graphe est résumé par le tableau ci-dessous, où les vecteurs figurent en lignes et les procédures en colonnes. Un "X" signifie que le vecteur garde son nom.

Nature	Occupation mémoire	Dico	DegPlus	TriTopo	Circuit
Fichier de base	dépend du logiciel	X			
Code	N*(6-8 alphanumériques)	X			X
Pointeur	N entiers	X	X	X	X
Liste	M entiers	X	X	X	X
NumTopo	N entiers		Dplus	X	MR
NumGraf	N entiers			X	Chemin

Si on se fixe un nombre moyen de 10 liaisons par tâche, dont 2 recouvrements ou COB, M=12N. Avec des entiers codés sur 2 octets (N<32000) et 1 alphanumérique par octet, il faut 38*N octets mémoires pour les données.

Pour la partie calcul proprement dit, le fichier des codes est inutile. Il peut être sauvegardé à la fin de ce traitement pour les éditions ultérieures. Le vecteur NumTopo ne servira que pour la prise en compte des COB. Pour éviter de garder ce vecteur, on peut remplacer en une seule lecture de Liste, Y par NumTopo(Y) lorsque Y est une COB.

Algorithme Remplace

```
But: remplace dans le vecteur Liste la COB (Y,X) par (Numtopo(Y), X)
Début
   I :=1
   tant que I ≤ M faire
      si Liste(I) >0 alors I := I+1
      sinon
         si Liste(I+1) < 0 alors Liste(I) := - NumTopo( -Liste(I)) finsi
         I := I+2
      finsi
   fintque
Fin
```

Phase de calcul

1 Initialisation des calculs.

Lorsque la phase de traitement du graphe seul est achevée, le graphe est complet et sans circuit, et l'on en connait une numérotation topologique. Le dictionnaire du graphe permet de prendre en compte les liaisons FD, DD et les COB. Il reste à intégrer les données chiffrées. Afin de limiter le nombre de vecteurs à garder en mémoire, on remarque que le fait de fixer une date minimale de début (resp maximale de fin) équivaut à imposer une borne minimale à la date de début au plus tôt (resp borne maximale à la date de fin au plus tard). De même, les dates impératives n'entrent jamais en concurrence avec les autres dates (imposées, plus tôt, plus tard). Ces dates peuvent donc être facilement prises en compte lors de l'initialisation de Dtot(X) et Ftard(X). Pour différencier dates imposées et dates impératives, les dernières seront affectées d'un signe négatif.

Le sommet Début (sommet 0), de durée nulle, a pour dates de début et fin (tôt et tard) la date de début du projet. Si certaines tâches ont réellement débuté, la date minimum de début sera prise comme date de démarrage du projet, sinon cette date sera demandée. Les tâches Z débutées ou achevées sont gardées dans le dictionnaire comme sommet possédant pour unique prédécesseur le sommet 0. D'où l'algorithme d'initialisation suivant

Algorithme Initialisation

Début

DebProjet := 0
pour les tâches X de 1 à N dans l'ordre où elles figurent dans le fichier de base
faire
Cal(X) := numéro du calendrier
Attente(X) := durée d'attente (0 par défaut)
Durée(X) := durée d'éxécution
Dtot(X) := date minimale de début imposée (0 par défaut)
Ftot(X) := ∞

```
        Dtard(X) := ∞
        Ftard(X) := date maximale de fin imposée ( ∞ par défaut)
        si X est débutée alors
           si DebProjet> date réelle de début alors DebProjet := date réelle de début
        finsi
         si X est débutée ou a un début impératif alors
           Dtot(X) := - date de début
           Dtard(X) := - date de début
        finsi
        si X est achevée ou a une fin impérative alors
           Ftard(X) := - date de fin
           Ftot(X) := - date de fin
        finsi
     finpour
     si DebProjet=0 alors
        écrire "donnez la date de démarrage du projet"
        lire DebProjet
     finsi
     Dtot(0) := DebProjet   Ftot(0) := DebProjet

  Fin
```

2 définition des procédures élémentaires de calcul.

Le calcul des dates au plus tôt d'une tâche X intervient dans le calcul général du projet hors COB et dans la prise en compte de celles-ci. On définira l'algorithme élémentaire suivant:

```
  Algorithme DatTot  (entrée: X, e/s: VectDeb, VectFin)

  But: VectDeb(Y) et VectFin(Y) contiennent  les dates au  plus tôt des Y de Pred(X);
       VectDeb(X) et VectFin(X) les valeurs initiales des dates de X
       on calcule les dates au  plus tôt définitives de X

  Début
     si VectFin(X)<0 alors X a une date impérative de fin, sortir(DatTot) finsi
     si VectDeb(X)<0 alors
        X a une date impérative de début
        VectFin(X) := Datcal(Cal(X), -VectDeb(X), Durée(X))
        sortir(DatTot)
     finsi
     dmax := VectDeb(X)
```

```
binf := pointeur (X)  bsup := pointeur (X + 1)
tant que binf < bsup faire
   Y := liste (binf)  dx := 0
   si Y >= 0 alors
      si duree (Y) = 0 alors
         dx := abs (VectDeb (Y)) + attente (Y)
      sinon
         dx := abs (VectFin (Y)) + attente (Y) + 1
      finsi
   sinon
      Y := -Y  binf := binf + 1  val := liste (binf)
   finsi
   si val > 0 alors
      si duree (Y) = 0 alors dx := abs (VectDeb (Y))
      sinon
         dx := Datcal (cal (Y), abs (VectDeb (Y)), val) + 1
      finsi
   finsi


   si dmax < dx alors dmax := dx
   binf := binf + 1
fintque

VectDeb(X) := Ouvre( cal(X), dmax, 1)
VectFin(X) := Datcal ( cal(X), VectDeb(X), Durée(X))

Fin
```

Le calcul des dates au plus tard de X intervient dans le calcul général du projet hors COB, dans la prise en compte de celles-ci, dans le calcul des dates libres. Selon le cas, on calcule les dates au plus tard de X impliquées par les dates de ses successeurs, soit les dates au plus tard, soit les dates au plus tôt. Par rapport à l'algorithme de base DatTard, on a à considérer les deux parties distinctes:

- calculer les dates correctes au plus tard de X dues à ses successeurs
- repercuter une date de X sur ses prédécesseurs.

Algorithme CalcDat (e: X s:Debut, Fin)

But: calcule les dates de début et fin au plus tard définitives de X

Début

```
si Fin<0 alors sortir( CalcDat) finsi
si Debut<0 alors
    Fin := Ouvre( Cal(X), Fin, -1)
    sortir( CalcDat)
finsi
dx := Datcal(cal(X), fin, -Durée(X) )
si Debut<dx alors Debut := dx finsi
Fin := Datcal ( cal(X), Debut, Durée(X))
```

Fin

Algorithme Repercute (e: X, Debut e/s: VectDeb, VectFin)

But: connaissant la date de début de X, répercute cette date sur VectDeb(Y) et VectFin(Y) pour tout Y de Pred(X).

Début

```
        § cas où X a une date imposée §

si Debut < 0 alors Debut := -Debut
binf := pointeur (X)  bsup := pointeur (X + 1)
tant que binf < bsup faire
    Y := liste (binf)
                    § cas liaison FD §
    si Y >= 0 alors
        ay := debut - attente (Y)
        si Durée (Y) > 0 alors ay := ay - 1
        si VectFin (Y) > ay alors VectFin (Y) := ay
                    § cas liaison DD §
    sinon

        Y := -Y binf := binf + 1  val := liste (binf)
        si val > 0 alors
            cy := datcal (cal (Y), debut - 1, -val)
            si VectDeb (Y) > cy alors VectDeb (Y) := cy
        finsi

        si val = 0 alors
            ay := debut
```

```
            si VectFin (Y) > ay alors VectFin (Y) := ay
          finsi
        finsi
        binf := binf + 1
    fintque

Fin
```

3 Calculs dus aux contraintes optionnelles de blocage.

Algorithme CobTot (e: I, J, Val s: modifie)

But: calcule les répercutions sur le début au plus tôt de I d'une COB (I,J) de valeur Val; le logique modifie est mis à vrai si la COB modifie le début de I.

```
Début
    Y := NumGraf(I)  X := NumGraf(J)
     dy := Ouvre( Cal(Y), Ftot(X) - Val  +1, 1)
    si Dtot(Y)<dy alors
        pour K de I à J
            Z := NumGraf(K)
            VectDeb(Z) := Dtard(Z)
            VectFin(Z) := Ftard(Z)
        finpour
    Repercut(X, Dtot(X), VecDeb, VecFin)
    pour K de J-1 à I par pas -1
        Z := NumGraf(K)
        CalcDat(Z, VecDeb(Z), VectFin(Z) )
        Repercut(Z, VecDeb(Z), VectDeb, VecFin)
    finpour
    CalcDat( Y, VecDeb(Y), VecFin(Y))
    si VecDeb(Y)<Y alors dy := VecDeb(Y) finsi
    si Dtard(Y)<dy alors dy := Dtard(Y)
    modifie := Dtot(Y) < dy
    si modifie alors
        Dtot(Y) := dy
        Ftot(Y) := DatCal( Cal(Y), dy, Duree(Y))
    finsi

Fin
```

Algorithme CobTard (e: I, J, Val s: modifie)

But: calcule les répercutions sur la fin au plus tard de J d'une COB (I,J) de valeur Val; le logique modifie est mis à vrai si la COB modifie la date au plus tard de J

```
Début
   Y := NumGraf(I)
   X := NumGraf(J)
   fx := Ouvre( Cal(X), Dtard(Y)+Val  -1, -1)
   si Ftard(YX)>fx alors
      pour K de I à J
         Z := NumGraf(K)
         VectDeb(Z) := Dtot(Z)
         VectFin(Z) := Ftot(Z)
      finpour
   pour K de I à J
      Z := NumGraf(K)
      DatTot(Z, VecDeb, VecFin)
   finpour
   si VecFin(X)>fx alors fx := VecFin(X)
   si Ftot(X)>fx alors fx := Ftot(X)
   modifie := Ftard(X) > fx
   si modifie alors  Ftard(X) := fx

Fin
```

Algorithme COB

But: prise en compte globale de toutes les COB. On rappelle que la COB (Y,X) figure dans le vecteur Liste sous la forme (J,X) avec J=NumTopo(Y)

```
Début
   recalcul := faux
   i := 1
   tant que i <= n faire
      jmin := ∞
      x := numgraf (i)
      si recalcul alors DatTot (x, dtot, ftot)
      binf := pointeur (x)  bsup := pointeur (x + 1)
      tant que binf < bsup faire
         y := liste (binf)
         si y < 0 alors
            binf := binf + 1  val := liste (binf)
            si val < 0 alors
               j:=-y
```

```
                CobTot (j, i, -val, modifie)
                si (modifie) alors si (j < jmin) alors
                    min := j
                    recalcul := vrai
                finsi
            finsi
        finsi
        binf := binf + 1
    finsi
    si jmin <>∞ alors
        i := jmin
    sinon
        i := i + 1
    finsi
finsi

si recalcul alors
    pour i de n à 1 par pas -1 faire
        x := numgraf (i)  CalcDat (x, dtard (x), ftard (x))
        Repercut (x, dtard (x), dtard, ftard)
        finpour
recalcul :=faux

pour i de n à 1 par pas -1 faire
    x := numgraf (i)
    binf := pointeur (x)  bsup := pointeur (x + 1)
    tant que binf < bsup faire
        y := liste (binf)
        si y < 0 alors
            binf := binf + 1  val := liste (binf)
            si val < 0 alors CobTard (-y, i, -val, modifie)
            si modifie alors recalcul := vrai
        finsi
        binf := binf + 1
    fintque

    si recalcul alors
        CalcDat (x, dtard (x), ftard (x))
        Repercut (x, dtard (x), dtard, ftard)
    finsi
finpour

Fin
```

4. Traitement principal.

Algorithme Ordon;

```
Début

   Dico(correct)
   si non CORRECT alors sortir (Ordon)
   TriTopo(VU)
   si VU>1 alors
      Circuits(VU)
      sortir (Ordon)
   finsi
   Initialisation

   § calcul des dates au  plus tôt sans les COB  §

   pour I de 1 à N
      X :=NumGraf(I)
      DatTot(X)
   finpour
   FinProj := Ftot(N)

   § calcul des dates au plus tard sans les COB  §

   Ftard(N) := - FinProj
   pour I de N à 1 par pas -1
      X :=NumGraf(I)
      CalcDat(X,Dtard(X),Ftard(X))
      Repercut(X, Dtard(X), Dtard, Ftard)
   finpour

   § Prise en compte des COB.  Calcul des dates libres §

   COB
   pour X de 1 à N
      Dlibre(X) := Dtard(X)
      Flibre(X) := Ftard(X)
   finpour
   pour I de 1 à N
      X := NumGraf(I)
      Repercut (X, Dtot(X), Dlibre, Flibre)
      CalcDat(X, Dlibre(X), Flibre(X))
   finpour

Fin
```

Les algorithmes précédents ont été programmés en Pascal sur un Zénith Z100. On donne ici quelques résultats obtenus à partir de variantes de l'exemple de départ.

1/ sans contraintes temporelles (début minimum ou imposé, fin maximum ou imposée), ni COB.

2/ avec introduction des contraintes temporelles, et sans les COB.

3/ avec contraintes temporelles et COB.

4/ en se plaçant en cours de projet

5/ avec introduction d'une date de fin imposée créant des tâches hypercritiques.

Tableau 1.

On tient compte ici uniquement des liaisons F-D et des recouvrements.

tâches	Dates au plus tôt		Dates au plus tard			Debut libre
Debut	-- 04/06	-- 04/06	-- 04/06	-- 04/06		
C1	lu 04/06	lu 11/06	ma 05/06	ma 12/06		lu 04/06
C2	lu 04/06	sa 23/06	je 07/06	ma 26/06		me 06/06
C3	lu 04/06	lu 18/06	je 21/06	je 05/07		me 20/06
C4	lu 04/06	lu 18/06	me 20/06	me 04/07		me 20/06
C5	lu 04/06	me 06/06	me 04/07	ve 06/07		lu 02/07
C6	lu 04/06	lu 11/06	di 08/07	di 15/07		di 08/07
G1	ma 12/06	je 14/06	me 13/06	ve 15/06		ma 12/06
G2	ve 15/06	me 20/06	lu 18/06	je 21/06		ve 15/06
G3	ma 26/06	ve 29/06	me 27/06	sa 30/06		ma 26/06
G4	lu 02/07	me 04/07	lu 02/07	me 04/07	C	
G5	je 05/07	lu 09/07	ve 06/07	ma 10/07		ve 06/07
G6	je 05/07	ma 10/07	je 05/07	ma 10/07	C	
G7	sa 30/06	lu 02/07	je 05/07	je 05/07		me 04/07
G8	lu 04/06	lu 04/06	ve 06/07	ve 06/07		me 04/07
A1	je 05/07	sa 07/07	sa 07/07	ma 10/07		sa 07/07
A2	me 11/07	ve 13/07	me 11/07	ve 13/07	C	
A3	ve 20/07	ve 20/07	ve 20/07	ve 20/07	C	
A4	me 18/07	je 19/07	me 18/07	je 19/07	C	
F1	ve 20/07	ma 24/07	ve 20/07	ma 24/07	C	
F2	me 11/07	ma 17/07	je 19/07	ma 24/07		je 19/07
Fin	me 25/07	me 25/07	me 25/07	me 25/07	C	

Il n'y a pas ici de véritable chemin critique du début à la fin du projet. Ceci tient au fait que le maçon ne travaillant pas le samedi, le début de G4 est reporté au lundi 2/7; ce qui permet d'achever G3 (unique prédécesseur de G4), soit le vendredi 29/6, soit le samedi 30/6.

Tableau 2.

Introduction des contraintes temporelles.

tâches	Dates au plus tôt		Dates au plus tard			Debut libre
Debut	-- 04/06	-- 04/06	-- 04/06	-- 04/06		
C1	lu 04/06	lu 11/06	me 06/06	me 13/06		lu 04/06
C2	lu 04/06	sa 23/06	sa 09/06	je 28/06		me 06/06
C3	lu 04/06	lu 18/06	di 24/06	di 08/07		di 24/06
C4	lu 04/06	lu 18/06	ve 22/06	ve 06/07		me 20/06
C5	lu 04/06	me 06/06	sa 07/07	lu 09/07		lu 02/07
C6	lu 04/06	lu 11/06	je 12/07	je 19/07		je 12/07
G1	ma 12/06	je 14/06	je 14/06	lu 18/06		ma 12/06
G2	ve 15/06	me 20/06	ma 19/06	ve 22/06		ve 15/06
G3	ma 26/06	ve 29/06	ve 29/06	ma 03/06		ma 26/06
G4	lu 02/07	me 04/07	me 04/07	ve 06/07		lu 02/07
G5	lu 09/07	je 12/07	lu 09/07	je 12/07	C	
G6	je 05/07	ma 10/07	sa 07/07	je 12/07		je 05/07
G7	sa 30/06	lu 02/07	ve 06/07	ve 06/07		ve 06/07
G8	-- 05/07	-- 05/07	-- 05/07	-- 05/07	C	
A1	ve 06/07	lu 09/07	ma 10/07	je 12/07		ma 10/07
A2	ve 13/07	ma 17/07	ve 13/07	ma 17/07	C	
A3	ve 20/07	ve 20/07	ve 20/07	ve 20/07	C	
A4	lu 23/07	ma 24/07	lu 23/07	ma 24/07	C	
F1	me 25/07	ve 27/07	me 25/07	ve 27/07	C	
F2	me 11/07	ma 17/07	lu 23/07	ve 27/07		lu 23/07
Fin	sa 28/07	sa 28/07	sa 28/07	sa 28/07	C	

La tâche G8 (date imposée) devient critique et a pour effet de reporter au 06/07 le début au plus tôt de A1.

Le début de G5 est reporté au 9 juillet et G5 devient critique.

La date de début minimale de F1 (le 20/07), est sans effet.

Tableau 3.

Résultats de l'exemple complet présenté au chapitre 1.

tâches	Dates au plus tôt		Dates au plus tard			Debut libre
Debut	-- 04/06	-- 04/06	-- 04/06	-- 04/06		
C1	lu 04/06	lu 11/06	me 06/06	me 13/06		me 06/06
C2	lu 04/06	sa 23/06	ve 08/06	me 27/06		ve 08/06
C3	lu 04/06	lu 18/06	di 24/06	di 08/07		di 24/06
C4	lu 04/06	lu 18/06	ve 22/06	ve 06/07		je 21/06
C5	lu 04/06	me 06/06	ma 03/07	je 05/07		ma 03/07
C6	sa 23/06	sa 30/07	je 12/07	je 19/07		je 12/07
G1	je 14/06	lu 18/06	je 14/06	lu 18/06	C	
G2	ma 19/06	ve 22/06	ma 19/06	ve 22/06	C	
G3	je 28/06	lu 02/07	je 28/06	lu 02/07	C	
G4	ma 03/07	je 05/07	ma 03/07	je 05/07	C	
G5	lu 09/07	je 12/07	lu 09/07	je 12/07	C	
G6	ve 06/07	me 11/07	sa 07/07	je 12/07		ve 06/07
G7	ma 03/07	ma 03/07	ve 06/07	ve 06/07		ve 06/07
G8	-- 05/07	-- 05/07	-- 05/07	-- 05/07	C	
A1	ve 06/07	lu 09/07	ve 06/07	lu 09/07	C	
A2	ve 13/07	ma 17/07	ve 13/07	ma 17/07	C	
A3	ve 20/07	ve 20/07	ve 20/07	ve 20/07	C	
A4	lu 23/07	ma 24/07	lu 23/07	ma 24/07	C	
F1	me 25/07	ve 27/07	me 25/07	ve 27/07	C	
F2	je 12/07	me 18/07	lu 23/07	ve 27/07		lu 23/07
Fin	sa 28/07	sa 28/07	sa 28/07	sa 28/07	C	

contrainte optionnelle entre les tâches		valeur demandee	valeur obtenue plus tôt	valeur obtenue plus tard
G3	C6	10	10	-9
A1	G1	1	26	26

Du fait qu'il existe un chemin de G1 à A1, la COB minimale entre ces deux tâches:
- au plus tôt bloque le début de G1 sur la fin de A1
- au plus tard bloque la fin de A1 sur le début de G1,

et rend ces deux tâches critiques.

En page 73, on trouvera le diagramme de Gantt correspondant (sortie sur imprimante).

Tableau 4

Nous nous plaçons ici en cours d'exécution du projet, au 20/06, en supposant que les délais du tableau 3, aient pu être respectés. Les tâches C1 et G1 sont achevées (dates de début et fin réelles), les tâches C2, C3, C4 et G2 sont débutées.

tâches	Dates au plus tôt		Dates au plus tard			Debut libre
Debut	-- 04/06	-- 04/06	-- 04/06	-- 04/06	-	
C1	-- 04/06	-- 12/06	-- 06/06	-- 12/06	-	
C2	-- 06/06	lu 25/06	-- 06/06	me 27/06		
C3	-- 06/06	me 20/06	-- 06/06	di 08/07		
C4	-- 20/06	me 04/07	-- 20/07	ve 06/07		
C5	lu 04/06	me 06/06	ma 03/07	je 05/07		ma 03/07
C6	sa 23/06	sa 30/07	je 12/07	je 19/07		je 12/07
G1	-- 14/06	-- 18/06	--14/06	-- 18/06	-	
G2	-- 19/06	ve 22/06	ma 19/06	ve 22/06	C	
G3	je 28/06	lu 02/07	je 28/06	lu 02/07	C	
G4	ma 03/07	je 05/07	ma 03/07	je 05/07	C	
G5	lu 09/07	je 12/07	lu 09/07	je 12/07	C	
G6	ve 06/07	me 11/07	sa 07/07	je 12/07		ve 06/07
G7	ma 03/07	ma 03/07	ve 06/07	ve 06/07		ve 06/07
G8	-- 05/07	-- 05/07	-- 05/07	-- 05/07	C	
A1	ve 06/07	lu 09/07	ve 06/07	lu 09/07	C	
A2	ve 13/07	ma 17/07	ve 13/07	ma 17/07	C	
A3	ve 20/07	ve 20/07	ve 20/07	ve 20/07	C	
A4	lu 23/07	ma 24/07	lu 23/07	ma 24/07	C	
F1	me 25/07	ve 27/07	me 25/07	ve 27/07	C	
F2	je 12/07	me 18/07	lu 23/07	ve 27/07		lu 23/07
Fin	sa 28/07	sa 28/07	sa 28/07	sa 28/07	C	

contrainte optionnelle entre les tâches		valeur demandee	valeur obtenue plus tôt	valeur obtenue plus tard
G3	C6	10	10	-9
A1	G1	1	26	26

Tableau 5.

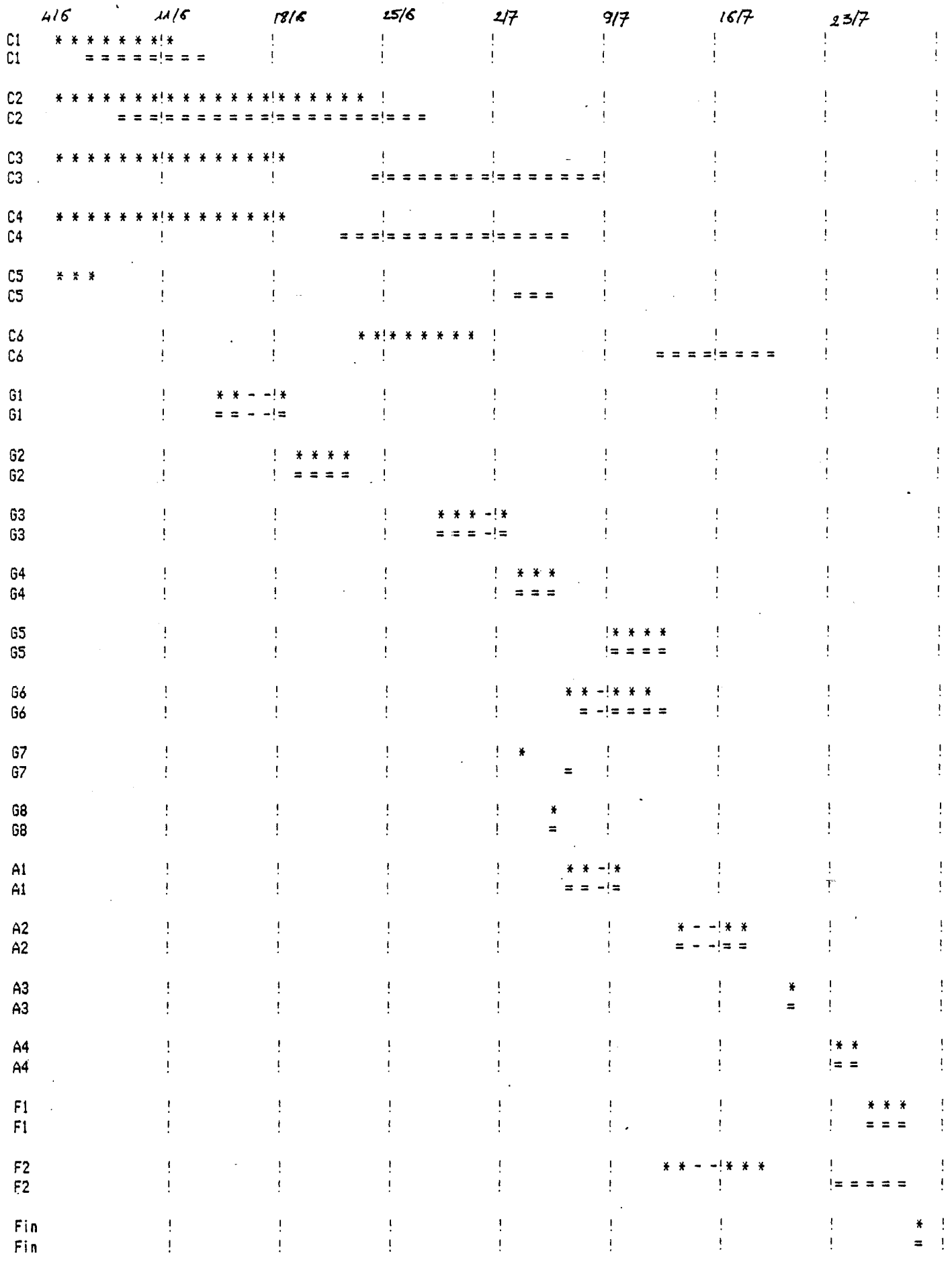
D'après les calculs (tableau 3), la date de fin au plus tard de F1 est le 27/07. On impose ici que cette date soit le 25/07.

tâches	Dates au plus tôt		Dates au plus tard			Debut libre
Debut	-- 04/06	-- 04/06	-- 04/06	-- 04/06		
C1	lu 04/06	lu 11/06	ma 05/06	ma 12/06		ma 05/06
C2	lu 04/06	sa 23/06	je 07/06	ma 26/06		je 07/06
C3	lu 04/06	lu 18/06	je 21/06	je 05/07		di 24/06
C4	lu 04/06	lu 18/06	me 20/06	me 04/07		me 20/06
C5	lu 04/06	me 06/06	ma 03/07	je 05/07		ma 03/07
C6	je 21/06	je 28/07	lu 09/07	lu 16/07		je 12/07
G1	me 13/06	ve 15/06	me 13/06	ve 15/06	C	
G2	lu 18/06	je 21/06	lu 18/06	je 21/06	C	
G3	me 27/06	ve 30/06	me 27/06	ve 30/06	C	
G4	lu 02/07	me 04/07	lu 02/07	me 04/07	C	
G5	lu 09/07	je 12/07	ve 06/07	ma 10/07	H	
G6	je 05/07	ma 10/07	je 05/07	ma 10/07	C	
G7	lu 02/07	lu 02/07	je 05/07	je 05/07		ve 06/07
G8	-- 05/07	-- 05/07	-- 05/07	-- 05/07	C	
A1	ve 06/07	lu 09/07	ve 06/07	lu 09/07	C	
A2	ve 13/07	ma 17/07	me 11/07	ve 13/07	H	
A3	ve 20/07	ve 20/07	ma 17/07	ma 17/07	H	
A4	lu 23/07	ma 24/07	je 19/07	ve 20/07	H	
F1	me 25/07	ve 27/07	lu 23/07	me 25/07	H	
F2	je 12/07	me 18/07	lu 23/07	ve 27/07		lu 23/07
Fin	sa 28/07	sa 28/07	sa 28/07	sa 28/07	C	

contrainte optionnelle entre les tâches		valeur demandee	valeur obtenue plus tôt	valeur obtenue plus tard
G3	C6	10	10	-8
A1	G1	1	27	27

Cette date de fin imposée sur F1 ne peut être respectée avec les données actuelles. Si l'on garde néanmoins cette date comme date de fin au plus tard, les calculs déterminent un certain nombre de tâches pour lesquelles les dates au plus tard sont inférieures aux dates au plus tôt. Ces tâches dites hypercritiques, sont celles dont les durées sont à réduire pour tenir la contrainte sur F1.

Les dates libres, calculées sur les dates au plus tôt, ne sont pas toutes significatives (G7).


4/6
11/6
18/6
25/6
2/7
9/7
16/7
23/7
C1
C1
C2
C2
C3
C3
C4
C4
C5
C5
C6
C6
G1
G1
G2
G2
G3
G3
G4
G4
G5
G5
G6
G6
G7
G7
G8
G8
A1
A1
A2
A2
A3
A3
A4
A4
F1
F1
F2
F2
Fin
Fin

Représentation graphique des résultats sur un nombre minimum de lignes.

Le diagramme de Gantt, ou diagramme à barres fut la première méthode mise au point par les responsables de planning. Elle a été élaborée vers 1900 par Gantt, un américain disciple de l'organisateur Taylor. Cette méthode permettait pour la première fois d'aborder les problèmes de délais, de répartition des tâches sous forme graphique. Elle sera la seule méthode jusqu'en 1958 (PERT etc...). Pour rappel, nous en donnerons ici une brève description.

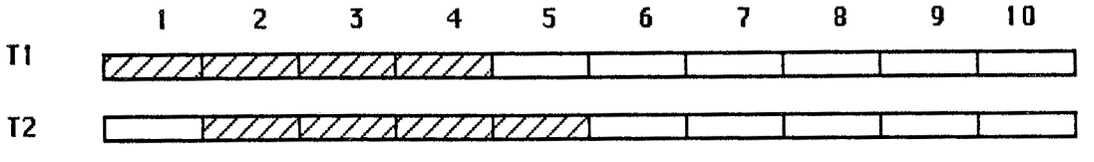
1.Présentation

La caractéristique fondamentale d'un graphique de Gantt est de représenter une tâche par un segment horizontal dont la longueur est proportionnelle à la durée de la tâche. Ces segments ou barres sont reportés sur un tableau quadrillé dans lequel chaque colonne correspond à une unité de temps. Dans la représentation de base,on ne représente qu'une seule tâche par ligne. Si une tâche est répétée sur plusieurs lignes on dira que la représentation est multi-ligne, uniligne sinon. Selon ce que l'on désire mettre en évidence, on peut définir plusieurs types de graphiques. Prenons par exemple un graphique représentatif de deux tâches T1 et T2, ayant comme dates au plus tôt, libres et au plus tard respectivement:

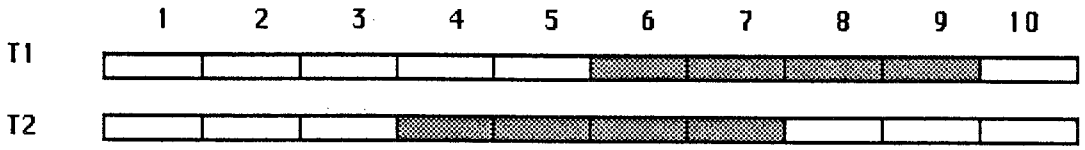
T1: 1à 4, 3 à 5, 6 à 9

T2: 2 à 5, 3 à 6, 4 à 7.

Variante 1: uniligne, avec représentation des dates au plus tôt:



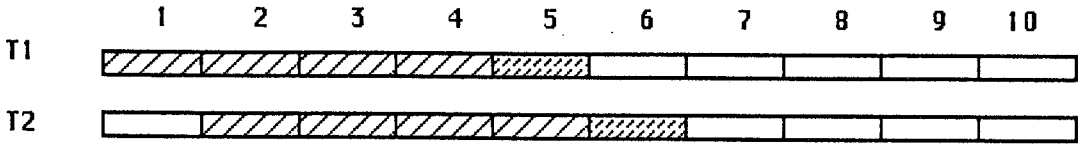
Variante 2: uniligne, avec représentation des dates au plus tard.



Variante 3: uniligne, avec représentation:

- des dates au plus tôt
- du début libre (si Dlibre(x) ≥ Ftot(x)) et de la fin libre (si Flibre(x)≥Ftot(x)).

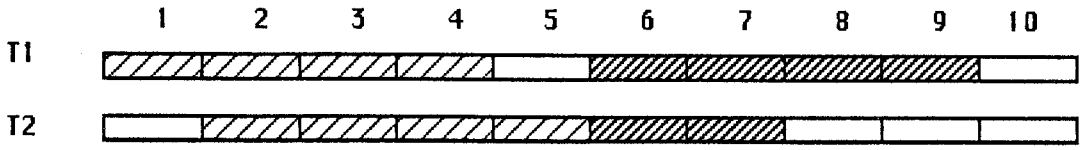
Ceci permet de visualiser la marge libre dans les problèmes sans calendrier.



Variante 4: uniligne, avec représentation:

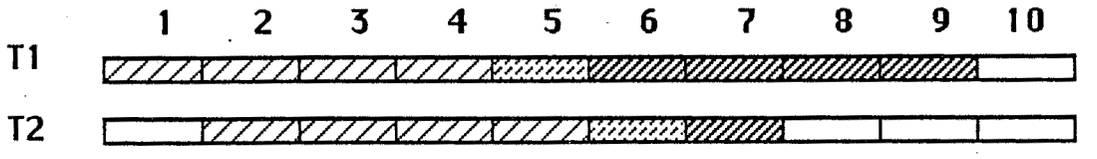
- des dates au plus tôt
- du début (si Dtard(x) ≥ Ftot(x)) et de la fin au plus tard (si Ftard(x)≥Ftot(x)),

pour visualiser la marge totale.

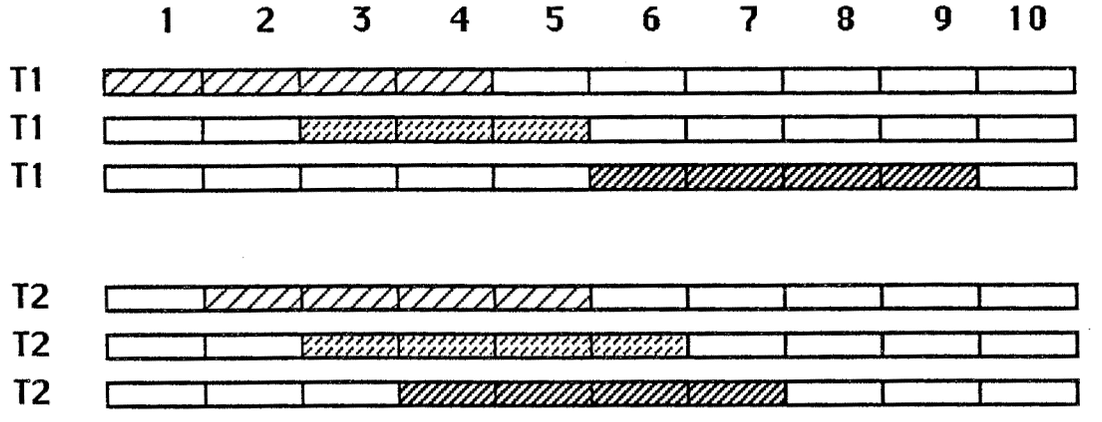


Variante 5: uniligne, avec représentation:

- des dates au plus tôt
- du début libre (si Dlibre(x) ≥ Ftot(x)) et de la fin libre (si Flibre(x)≥Ftot(x))
- du début au plus tard (si Dtarde(x) ≥ Ftot(x)) et de la fin l(si Ftard(x)≥Ftot(x)).



Variante 6: sur 3 lignes donnant deux ou trois groupes de dates.



Dans les problèmes avec calendrier, cette variante est la seule permettant de visualiser les marges libres et totales.

Ces graphiques présentent de nombreux avantages:

- ils montrent clairement la progression du travail
- le tracé (en pointillé par exemple) d'une barre représentant le travail réellement accompli, permet de suivre à tout moment l'avancement des travaux.

Par contre, leur inconvénient le plus grave, est qu'ils ne montrent pas les relations entre tâches. Cependant les qualités de clarté et de simplicité de cette représentation sont telles qu'elle reste une demande des praticiens et demeure un complément utile au PERT.

Une modification évidente pour une meilleure lisibilité du diagramme sur de grands projets, est de placer plusieurs tâches par ligne. Par la suite, nous nous intéresserons uniquement à ce type de représentation condensée et notre problème sera de minimiser le nombre de lignes nécessaires. Pour des raisons de clarté, nous imposerons de plus que dans les représentations multi-lignes, les tâches d'un

ensemble de lignes liées suivent le même ordre.

Deux types de diagrammes peuvent être envisagés:

- des représentations basées sur le seul réseau, indépendemment des durées et résultats. Les tâches étant toujours rangées de manière identique, ceci permet de bien visualiser le glissement des tâches dans le temps lorsque les durées varient.
- des représentations liées aux résultats.

2. Vocabulaire.

Nous allons tout d'abord rappeller certaines définitions et propriétés de théorie des graphes utilisées par la suite.

Un graphe G non orienté est dit triangulé si tout cycle de longueur > 3 admet une corde (c'est à dire une arête reliant deux sommets non consécutifs du graphe).

Un graphe simple G=(X,E) est appellé graphe de comparabilité s'il est possible, en orientant les arêtes d'en faire le graphe (X,U) d'une relation d'ordre, c'est à dire avec:

$(x,y) \in U,\ (y,z) \in U \Rightarrow (x,z) \in U$ (transitivité)

$(x,y) \in U \Rightarrow (y,x) \in U$ (antisymétrie).

Soit P = {1,2,3...N } un ensemble fini de cardinalité N muni d'une relation de préordre notée "<". Associons à P le graphe orienté $\mathcal{C}$ défini de la manière suivante: les sommets $\{X_1, X_2 \ldots X_N\}$ sont en bijection avec {1,2,3...N } et il existe un arc (Xi,Xj) si i<j. Le graphe obtenu en supprimant l'orientation est triangulé et de comparabilité.

Soit $\mathcal{C}$ une famille de n intervalles fermés: Ti = { di, fi } pour i=1,2..., n. On appelle graphe représentatif de la famille $\mathcal{C}$ (ou graphe d'intervalles) le graphe $\mathcal{R}$formé en associant respectivement aux sommets A1, A2,..., An du graphe les intervalles T1, T2,...,Tn ; deux sommets étant joints si et seulement si les intervalles correspondants s'interceptent. Un tel graphe est triangulé.

De plus (Gilmore et Hoffman 1964), un graphe simple G est un graphe d'intervalle si et seulement si, il est triangulé et si son complémentaire est un graphe de comparabilité.

Pour un graphe simple G, on désigne par:

α (G) le nombre de stabilité,

θ (G) le nombre minimum de cliques qui partitionnent X,

γ (G) le nombre chromatique,

ω (G) le nombre maximum de sommets qui forment une clique.

Un graphe sera dit α-parfait si l'on a

$$\alpha (G_A) = \theta (G_A) \qquad (A \subset X).$$

Un graphe sera dit γ-parfait si l'on a

$$\gamma (G_A) = \omega (G_A) \qquad (A \subset X).$$

Lovasz [LO] a démontré qu'un graphe est α-parfait si et seulement si il est γ-parfait (théorème des graphes parfaits). Les graphes de comparabilité, les graphes triangulés et les graphes d'intervalles sont parfaits.

3 Représentation basée sur le réseau.

Cette représentation doit être indépendante des temps plaqués ultérieurement sur les tâches. Ceci implique que, quelles que soient deux tâches consécutives X et Y représentées sur une ligne, X soit achevée avant le début de Y. Une condition nécessaire et suffisante, est que ces tâches soient liées par une relation de descendance de type fin X, début Y. On ne pourra donc pas utiliser cette représentation dans le cas uniligne avec dates au plus tôt et plus tard (4 et 5). Si les recouvrements ne sont pas trop nombreux, le découpage en lignes se fera en tenant compte des seules liaisons FD. Dans le cas contraire, cette représentation n'est pas adaptée. Par la suite, nous considérerons uniquement des graphes où les relations sont de type FD. On supposera de plus que les sommets sont numérotés selon un tri

sont de type FD. On supposera de plus que les sommets sont numérotés selon un tri topologique et on notera $\mathcal{G}$= (X, E) le graphe après renumérotation.

La relation de descendance est une relation de préordre. Nous retrouvons là le problème de décomposition de Dilworth [DI]. Soit P= {1,2,3...N } un ensemble fini de cardinalité N muni d'une relation de préordre notée "<". Deux éléments i et j tels que l'on ait ni i<j, ni j<i, seront dit incomparables. Un sous ordre de P est un sous ensemble de P de un ou plusieurs éléments X1,X2,...,Xq totalement ordonné:

$$X1 < X2 < ... < Xq .$$

Une décomposition de P est une partition de P en sous ordres (il existe au moins la décomposition triviale en N sous ordres à 1 élément). On recherche une décomposition ayant un nombre minimum de sous ordres.

Si nous associons à P le graphe orienté $\mathcal{G}$ défini précédemment, la notion de sous ordre coincide avec la notion de chaîne dans un graphe, excepté le fait qu'un sommet unique constitue une chaîne de longueur nulle. Une décomposition de P est une partition de $\mathcal{G}$ en chaînes. Deux éléments incomparables ne pouvant appartenir à un même sous ordre, le cardinal d'un ensemble d'éléments incomparables deux à deux est inférieur ou égal au nombre de sous ordres d'une décomposition.

Le théorème de Dillworth établit qu'il existe toujours un ensemble de sommets deux à deux incomparables et une partition de même cardinalité, et par suite un ensemble maximal de sommets incomparables et une décomposition minimale.

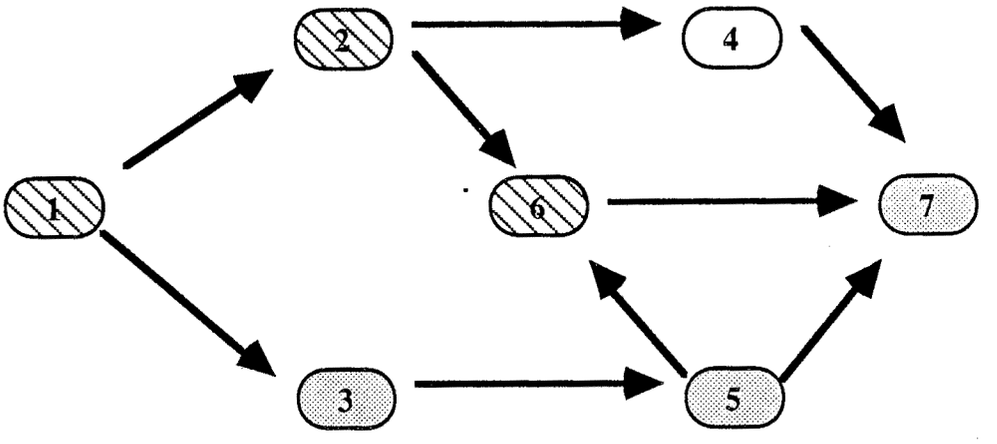
De nombreuses démonstrations de ce théorème existent dans la littérature: Dilworth (1950), Fulkerson (1956), Tverberg (1967), Trotter (1975), Pretzel (1979). Pour construire notre algorithme, nous partirons de la construction due à Fulkerson [FU]. Etant donné P, on construit le graphe biparti B = (G,D; E) à 2N sommets de la manière suivante:

- à tout élément i de P on associe un sommet gi de G et un sommet di de D
- on crée une arête (gi,dj) ssi i<j dans P

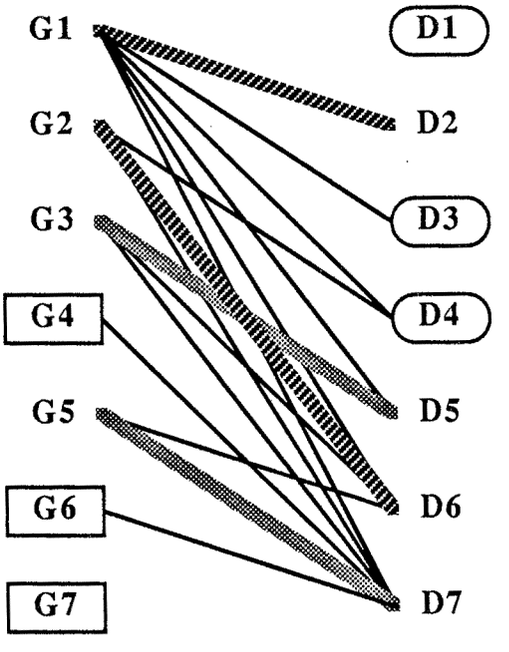
A tout couplage dans B correspond une décomposition de P et au couplage

maximum correspond une décomposition en un nombre minimum de chaînes.

Par exemple, si l'on considère le graphe d'ordonnancement suivant que l'on a décomposé en: { (1, 2, 6), (4), (3, 5, 7) }



on obtient le graphe biparti B et le couplage suivant:



Un sommet i est l'extrémité initiale (resp terminale) d'une chaîne ssi di (resp gi) est non saturé par le couplage. Par suite, un couplage sera maximum s'il n'existe pas de chaîne alternée joignant un sommet non saturé de G à un sommet non saturé de D. Dans l'exemple, il existe deux chaînes alternées:

G4, D7, G5, D6, G2, D4 conduisant à la décomposition: { (1, 2, 4, 7) , (3, 5, 6) }

G6, D7, G5, D6, G2, D4 conduisant à la décomposition: { (1, 2, 4) , (3, 5, 6, 7) }.

D'un point de vue pratique, le but est de construire un algorithme permettant de simuler la recherche d'une chaîne alternée dans B, en utilisant uniquement le graphe d'ordonnancement de départ © = (X, E)

- sans dupliquer les sommets
- sans ajouter d'arcs au graphe.

On se fixe comme objectif supplémentaire de limiter la place mémoire utilisée. L'optimum serait de ne pouvoir travailler qu'avec le dictionnaire du graphe et un vecteur mémorisant les chemins. Ceci semble impossible, et, dans la solution trouvée, on doit utiliser un vecteur supplémentaire pour les calculs intermédiaires.

Préliminaire

Soit X et Y deux sommets de B, on posera L(X,Y) la longueur minimale des chaînes alternées les joignant , s'il en existe. Soit d un sommet droit non saturé. Une des manières de trouver une chaîne alternée augmentante d'origine d, est de déterminer successivement les ensembles V_1, W_1, V_2 , $W_2 \ldots W_{p-1}$,V_p des sommets de B tels que:

- $V_i = \{ x \in G \ / \ L(d,x) = 2i - 1 \}$

- $W_i = \{ x \in D \ / \ L(d,x) = 2i \}$,

et de s'arrêter:

- soit parce que W_{p-1} ou V_p est vide; dans ce cas, il est inutile, dans une recherche ultérieure, de reconsidérer les sommets $x \in V_1 \cup W_1 \ldots W_{p-1} \cup V_p$;

- soit parce que V_p contient un sommet gauche non saturé; dans ce cas, on a trouvé une chaîne augmentante.

Soit une décomposition en ligne du graphe ©. Une telle décomposition existe toujours; par exemple on partira de la décomposition triviale d'une tâche par ligne. On associera à chaque sommet x de ©, une marque Mx:

- Mx < 0 si X est abandonné
- Mx = 0 si X non encore exploré
- Mx > 0 si X a été exploré, avec
  - Mx = X si Dx est dans W_{i-1}
  - Mx = Y si Gx a été marqué dans B par Dy

Le point délicat de ce marquage est le fait qu'un sommet x de © n'a qu'une marque, mais que les deux sommets correspondants de B peuvent appartenir l'un à Vi , l'autre à Wj.

cas 1: $Dx \in V_i$ et $Gx \in W_j$ avec $i \leq j$

au pas i, X a été marqué parce qu'il existait un chemin de X à un sommet Y, tel que $Dy \in W_i$. Il en est de même pour tous les prédécesseurs de X. Par suite, V_{j+1} sera identique, qu'il soit construit sur W_j ou sur W_j - {Dx}. La marque de X est conservée.

cas 2: $Dx \in V_i$ et $Gx \in W_j$ avec $j < i$

lorsque W_{i+1} a été déterminé, X doit pouvoir être marqué une deuxième fois au pas j; on doit remettre sa marque à zéro. L'algorithme complet sera alors:

initialement on a une tâche par ligne; les marques sont à zéro.

considérer les sommets début de lignes d, par ordre croissant:

1. poser Md=d
2. tant qu'il existe un sommet Y tel que My=Y faire
   marquer les sommets X non encore explorés (Mx=0) et tels qu'il existe un chemin de X à Y.
   poser My=0
   marquer les sommets Z non encore explorés, tels que X vient d'être marqué, et X précède Z dans la décomposition courante (Dz est dans W_i).
   si aucun sommet X ou aucun sommet Z n'a été trouvé
      abandonner tous les sommets explorés (si Mx>0, poser Mx=-Mx)
   finsi.
   si un sommet X fin de ligne a été marqué, réduire le nombre de lignes.
fintque.

Implémentation

En pratique, on travaille sur le graphe d'ordonnancement initial et non sur ©. On suppose que l'on connait un tri topologique du graphe de départ (cf chap 1). Le vecteur NumGraf (I) donnera le sommet du graphe en Ième position dans le tri (autrement dit I dans ©). La procédure Liste(X, Dgm, P) donne dans P les Dgm prédécesseurs du sommet X. On utilisera les 2 vecteurs:

vecteur Suivant(X) qui donne le successeur de X dans la ligne
- Suivant(X)=Y si X<Y
- Suivant(X)=X si X est une extrémité terminale

vecteur Marq:
contient le marquage précédemment défini.

Afin de limiter le champ des calculs, Mini et Maxi donne l'intervalle des sommets à prendre en compte.

Algorithme Marquage (s: Mini, Maxi, Sfin, Mieux, Arret)

But: connaissant W_{i-1} marquer les sommets Y tels qu'il existe un chemin de Y à X pour déterminer V_i et W_i.

En entrée, on dispose d'au moins un sommet X dans W_i-1 (Mini≤X≤Maxi et Marq(X)=X). On considère ces X par ordre décroissant. Par marquage arrière, on marque X les sommets Y non encore marqués et tel qu'il existe un chemin de Y à X. Le graphe étant topologique, ceci se fait en une seule passe. Si Y est une extrémité terminale (Suivant(Y)=Y) il y a une amélioration (Sfin=Y, Mieux:=vrai et Arret=vrai) sinon soit Z=Suivant(Y) avec obligatoirement Z>Y: si Z n'est pas encore exploré (Marq(Z)=0), Z ne peut plus être marqué (les X sont pris par ordre décroissant) et Z appartient à Vi+1 (Arret := faux)

```
début
   M1 := Mini   M2 := Maxi   Arret := vrai
   pour I de M2 à M1par pas -1 faire
      X := NumGraf(I)
      si Marq(X) := X alors
         S := X
         tant que S≥Mini faire
            si Marq(S)=X alors
               Liste( X, DGM, P)
               pour J de 1 à DGM faire
                  Y := P(J)
                  si Suivant(Y)=Y alors
                     Arret := vrai  Mieux := vrai  Sfin := Y  Marq(Y) := X
                     si Y<Mini alors Mini := Y
                  finsi
                  si Mieux alors sortir(Marquage)
                  si Marq(Y)=0 alors
                     Marq(Y) := X   Z := Suivant(Y)
                     si Marq(Z)=0 alors
                           Marq(Z)=Z Arret := faux
                           si Z>Maxi alors Maxi := Z
                     finsi
                     si Y<Mini alors Mini := Y
                  finsi
               finpour
            finsi
            S := S-1
         fintq
         Marq(X) := 0
      finsi
   finpour
fin
```

Algorithme Dilworth:

But: détermine une décomposition en un nombre minimum de ligness

```
Début

§ initialisation §

   pour X de 1 à N faire
      Suivant(X) := X
      Marq(X) := X
   finpour

   pour I de 2 à N faire
      X := NumGraf(I)
      Marq(X) := X  Mini := X
      Maxi := X    Z:= X
      Mieux := faux  Arret := faux
      répéter
         Marquage (Mini, Maxi, Sfin, Mieux, Arret)
      jusqu'à Arret
      si Mieux alors
         répéter
            Z := Marq(Sfin)  S := 1
            tant que Suiv(S) ≠ Z faire S := S+1
            Suivant(Sfin) := S   CH(Z) := Sfin
            Sfin := S
         jusqu'à X=Z
         pour Y de Mini à Maxi faire
            si Marq(Y)>0 alors Marq(Y)=0
         finpour
      finsi
      si non Mieux alors
         pour Y de Mini à Maxi faire
            si Marq(Y)>0 alors Marq(Y) := - Marq(Y)
         finpour
      finsi
   finpour

Fin
```

4. Représentation liée aux résultats.

Ici on connait les résultats de l'ordonnancement. L'intervalle nécessaire pour représenter une tâche est parfaitement défini et, selon la variante choisie, est donné par une date de début et une date de fin.

Dans le cas de l'uniligne, le choix de ces dates est évident. Pour le multi-ligne, si l'on exige de retrouver les tâches dans le même ordre sur les lignes liées, on prendra le minimum des dates au plus tôt à représenter et le maximum des dates au plus tard. Dans la variante 6, où l'on veut représenter sur 3 lignes distinctes dates au plus tôt, dates libres et dates au plus tard,on gardera comme bornes de l'intervalle, Dtot et Ftard.

Dans cette représentation, deux tâches pourront figurer sur une même ligne si les intervalles de temps qu'elles occupent ne s'interceptent pas. Le problème de la représentation en un nombre minimum de lignes, se ramène ainsi au problème général de la coloration d'une famille $\mathfrak{C}$ de n intervalles fermés

Ti = { di, fi } pour i=1,2..., n.

Du fait de la relation entre graphe d'intervalle et graphe de comparabilité, il est possible de ramener ce problème au précédent. Considérons la relation de préordre suivante sur $\mathfrak{C}$: Ti < Tj $\Leftrightarrow$ fi < dj. On vérifie aisément qu'une décomposition de $\mathfrak{C}$ muni de cette relation donne une coloration. Le théorème de Dilworth permet là aussi d'affirmer qu'il existe une décomposition minimum; pour la déterminer , il suffit d'appliquer l'algorithme du paragraphe précédent sur le graphe associé au préordre. Ceci implique les deux modifications suivantes sur le vecteur NumGraf et le sous traitement Liste:

- Trier les tâches par dx croissant, et à dx égaux, par fx croissants (ce qui donne un tri topologique des intervalles): NumGraf(I) donne la tâche X en Ième position dans ce tri.

- Le sous traitement Liste(e: X, s: Dgm, P) donnera dans P les Dgm tâches Y telles que fy < dx.

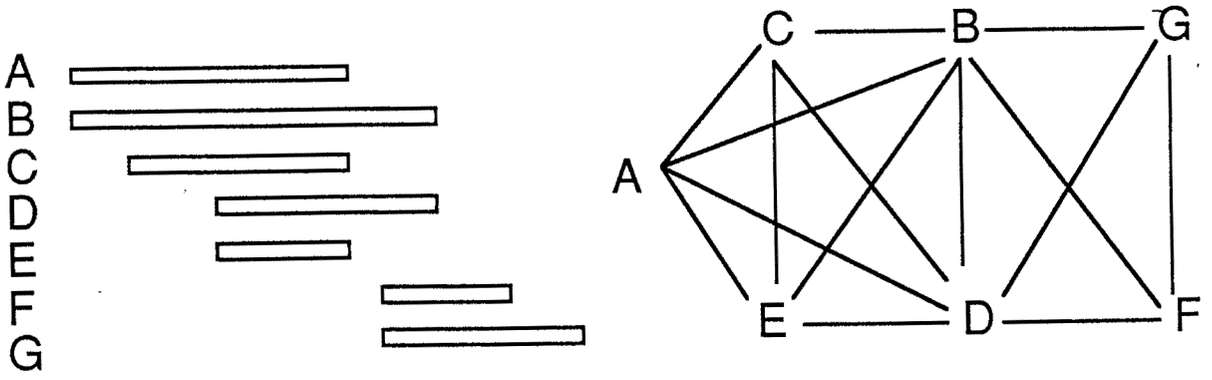
Une autre approche possible, est de considérer le graphe représentatif de la famille $\mathfrak{C}$. Dans la littérature (cf Golumbic [GO]), on trouve des algorithmes polynomiaux pour colorier un graphe triangulé, et par conséquent un graphe d'intervalles, avec un nombre minimum de couleurs (problème qui est NP-complet pour les graphes généraux). Ces algorithmes se fondent sur les points suivants:

- on appelle sommet simplicial un sommet x tel que le graphe induit par ses adjacents soit une clique;
- tout graphe triangulé admet un sommet simplicial (Dirac 1961);
- tout sous graphe d'un graphe triangulé étant lui-même triangulé, il est possible de renuméroter V1, V2...Vn les sommets de G de telle manière que le graphe induit par Vi, Vi+1... Vn soit triangulé et que Vi soit simplicial pour ce graphe. Une telle renumérotation est dite aussi schéma parfait (perfect scheme ou perfect vertex elimination scheme). Ce schéma déterminé, on affecte à V1 la couleur 1, à Vi la plus petite couleur non affectée aux adjacents de Vi. Un schéma parfait s'obtient en temps polynomial par l'algorithme:

- marquer 0 tous les sommets
- pour i de n à 1 par pas -1
  choisir un sommet x de marque maximale;
  renuméroter i ce sommet;
  ajouter 1 à la marque des adjacents de x non encore renumérotés.

fpour

Par exemple, si nous considérons le graphe d'intervalle:



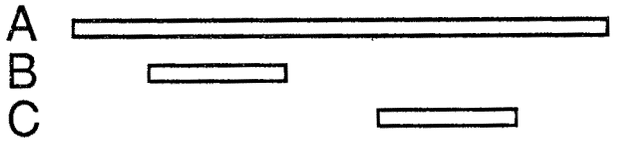
un des schémas parfaits est A C E G B D F, et l'on obtient la coloration:

A: 1, C:2, E:3, G:1, B:4, D:5, F:2

Dans notre cas, les données initiales sont les bornes des intervalles. Déterminer l'ensemble des adjacents d'un intervalle donné, demanderait des comparaisons entre les bornes. Le théorème suivant permet de les éviter pour établir le schéma:

théorème: soit une famille d'intervalles Ti = {di,fi} rangés par fi croissant; alors la suite A1, A2...An des sommets correspondants est un schéma parfait du graphe représentatif.

En effet, au pas i, les sommets A1...Ai-1 ont été éliminés; par construction Ti est l'intervalle de fi minimum et si Tk (k=i+1...n) intercepte Ti, Tk est tel que $dk \leq fi \leq fk$. Ces intervalles ayant au moins la valeur fi en commun, les sommets Ak représentatifs engendrent une clique. On montrerait de même que le rangement par di décroissant induit un schéma parfait. Par contre, on vérifie sur le contre exemple suivant que le rangement par di croissant (ou fi décroissant), n'en constitue pas un.



La recherche des adjacents peut aussi être évitée dans la phase de coloration en gardant, pour chacune des couleurs, la borne supérieure du dernier intervalle colorié par cette couleur. D'où l'implémentation:

<u>implémentation</u>

Les vecteurs Debut et Fin donnent les bornes de l'intervalle, Ordre(I) donne le sommet X en ième position après rangement des intervalles par Fin(X) croissant. Le vecteur Limite(C) donne la borne supérieure de la couleur C. Le vecteur Couleur(X) donne la couleur de X.

```
Algorithme Coloration ( e: Début, Fin, Ordre)

début
   pour X de 1 à N faire Limite (X):= 0
   pour I de 1 à N faire
      X :=  Ordre (I)
      C:=0
      répéter
         C := C+1
      jusqu'à Limite(C) < Debut(X)
      Couleur (X)=C
      Limite(C) := Fin(X)
   Finpour

Fin
```

Cet algorithme demande de n examens du vecteur Limite (les intervalles se succèdent) à $n^2/2$ examens (les intervalles se chevauchent).

L'algorithme généralement cité pour la coloration d'une famille d'intervalles (cf [GM]) n'a pas pour base les graphes triangulés et se justifie par récurrence. On suppose qu'à l'étape courante, un certain nombre d'intervalles aient déjà été coloriés avec les couleurs 1,2...c-1. On applique l'algorithme:

sélectionner l'intervalle non colorié tel que di soit minimum. Soit T_{i1} cet intervalle. Lui affecter la couleur c. Déterminer l'intervalle Ti2={di2,fi2} non colorié tel que di2>fi1 et di2 minimum. Lui affecter la couleur c et continuer le processus jusqu'à ce qu'on ne puisse plus colorier d'intervalle avec la couleur c. Passer à la couleur c+1 jusqu'à ce que tous les intervalles aient été coloriés.

Cet algorithme demande $p^2/2$ examens de chaque intervalle (chaque fois qu'un intervalle est choisi, il faut examiner les p-1 restants). Son efficacité moyenne peut être améliorée si les intervalles sont ordonnés par di croissant. L'algorithme devient alors :

soit Ti1 le premier intervalle non coloré

soit Ti2 le premier intervalle non coloré suivant i1 tel que di2<fi1.

L'efficacité de cette variante dépend alors des données de départ, variant de p à

$p^2/2$ examens de chaque intervalle; dans le meilleur des cas, les intervalles sont consécutifs, dans le pire ils se chevauchent tous. Cet algorithme, bien que présentant quelques similitudes avec le précédent, est fondamentalement distinct, puisque le tri par di croissant ne constitue pas un schéma parfait. On peut remarquer aussi qu'ils ont une efficacité similaire.

implémentation

Le vecteur Ordre(I) donne le sommet X en ième position après rangement des intervalles par Debut(X) croissant.

```
Algorithme Color-bis(Début, Fin, Ordre)

début
   pour X de 1 à N faire Couleur (X):= 0
   C := 1
   pour I de 1 à N faire
      X :=  Ordre (I)
      si Couleur (X)=0 alors
         Couleur (X) := C     M := Fin(X)
         pour J de I+1 à N faire
            X :=  Ordre (J)
            si Couleur(X)=0 et Début  (X)>M alors
               Couleur(X) := C
               M := Fin(X)
            Finsi
         Finpour
         C := C+1
      Finsi
   Finpour

Fin
```

L'ordonnancement du projet représenté par le graphe de la figure 1, a été calculé pour deux jeux de durées des tâches. Le tableau 1 donne les résultats. A titre d'illustration, les pages suivantes donnent les représentations:

- unilignes des dates au plus tôt
- multi-lignes des dates au plus tôt et au plus tard.

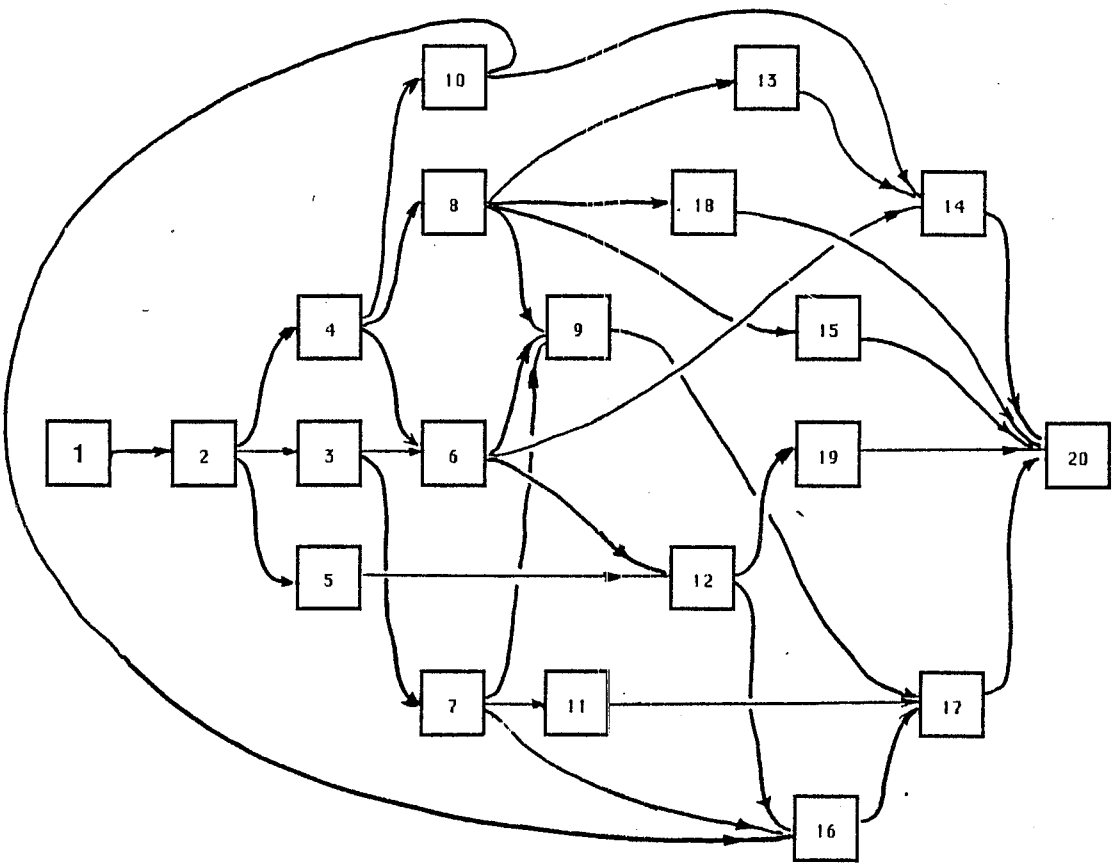
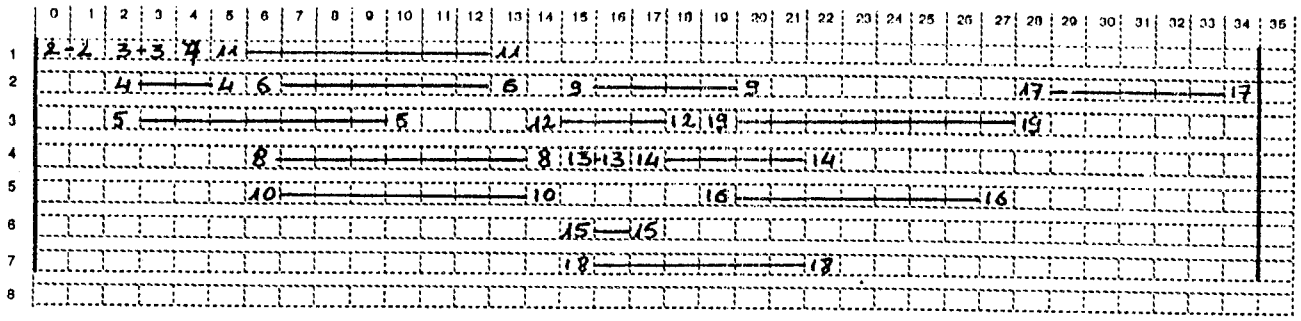


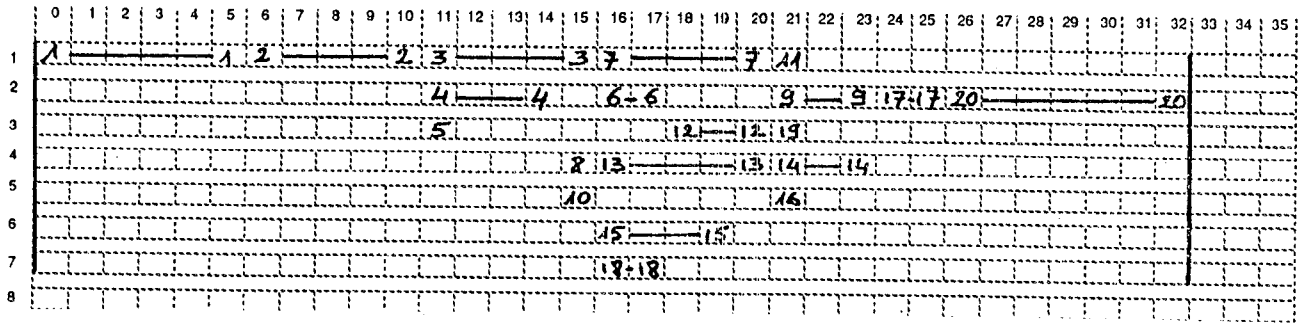
Figure 1

	jeu d'essai 1					jeu d'essai 2				
	durée	plus tot		plus tard		durée	plus tot		plus tard	
1	0	0	0	0	0	6	0	5	0	5
2	2	0	1	0	1	5	6	10	6	10
3	2	2	3	4	5	5	11	15	11	15
4	4	2	5	2	5	4	11	14	13	16
5	9	2	10	5	13	1	11	11	19	19
6	8	6	13	6	13	2	16	17	18	19
7	1	4	4	18	18	5	16	20	16	20
8	9	6	14	13	21	1	15	15	17	17
9	6	15	20	22	27	3	21	23	21	23
10	9	6	14	10	18	1	15	15	22	22
11	9	5	13	19	27	1	21	21	23	23
12	5	14	18	14	18	3	18	20	20	22
13	2	15	16	27	28	5	16	20	18	22
14	6	17	22	29	34	3	21	23	23	25
15	3	15	17	32	34	4	16	19	22	25
16	9	19	27	19	27	1	21	21	23	23
17	7	28	34	28	34	2	24	25	24	25
18	8	15	22	27	34	2	16	17	24	25
19	10	19	28	25	34	1	21	21	25	25
20	0	35	35	35	35	6	26	32	26	32

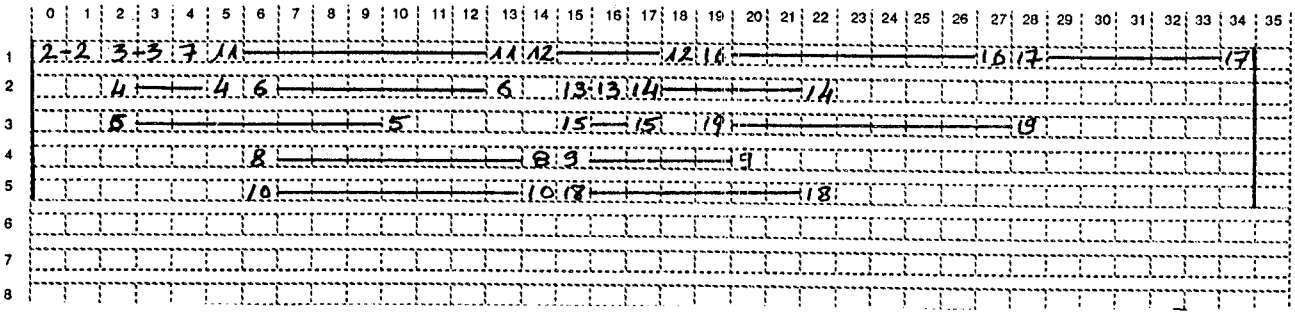
Tableau 1

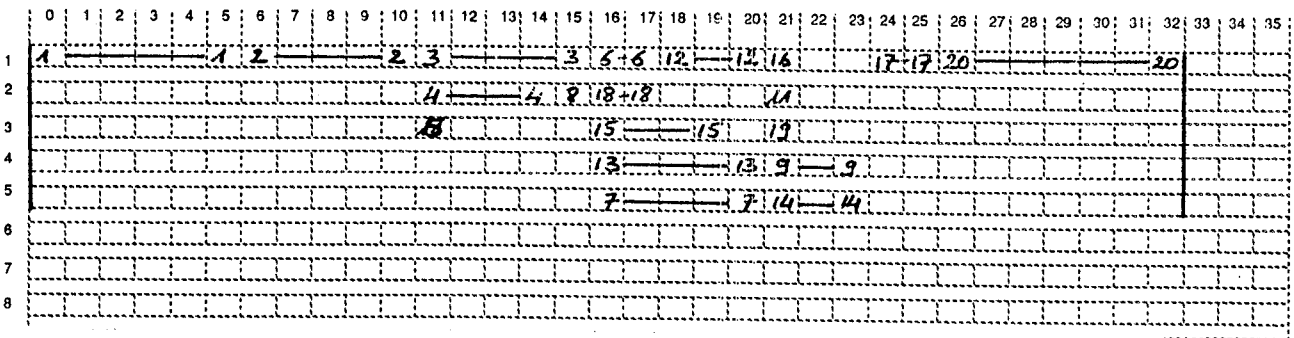
Représentation basée sur le réseau.



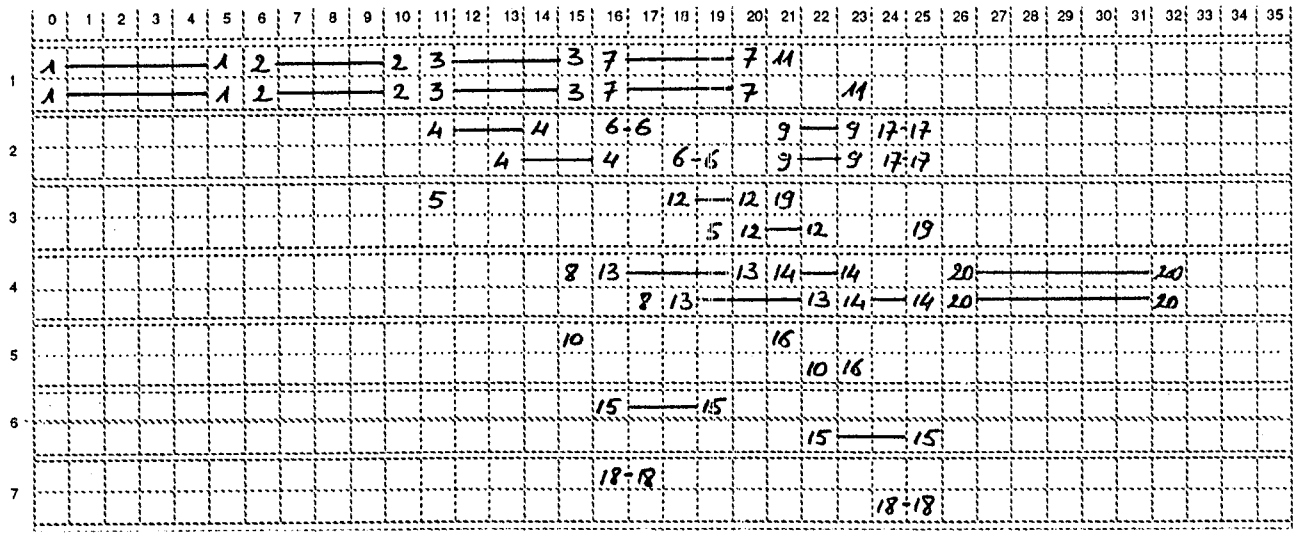


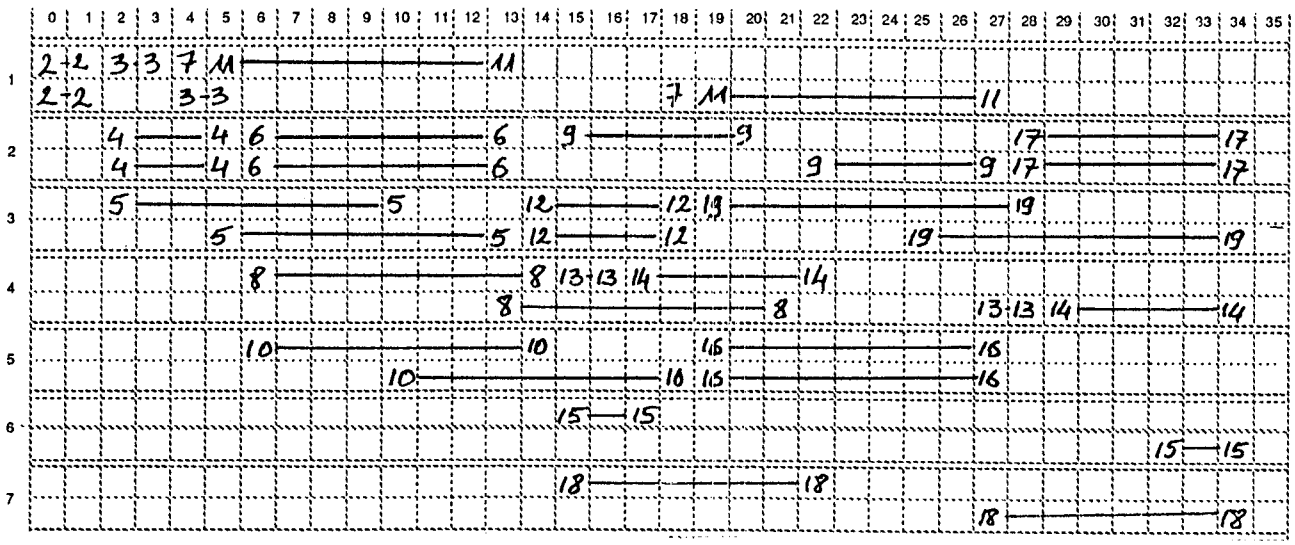
Représentation basée sur les résultats.



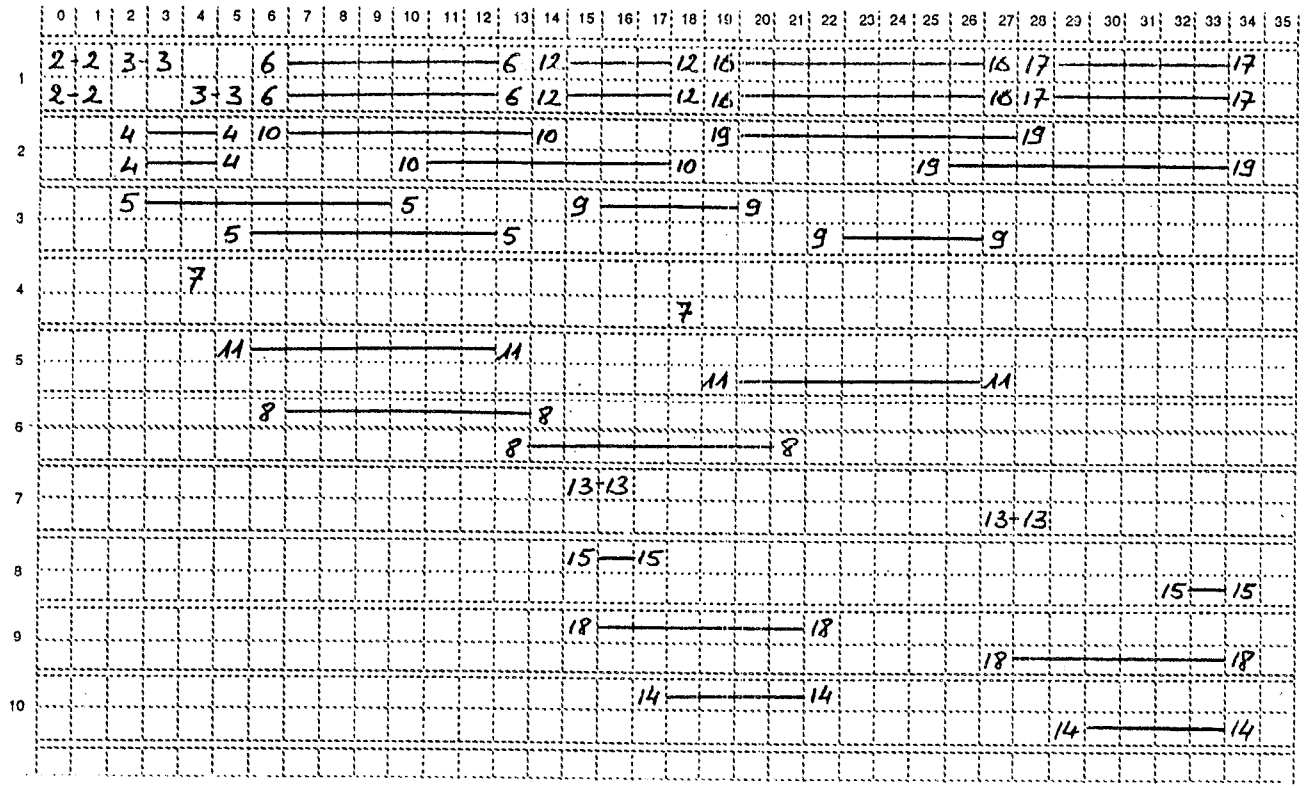


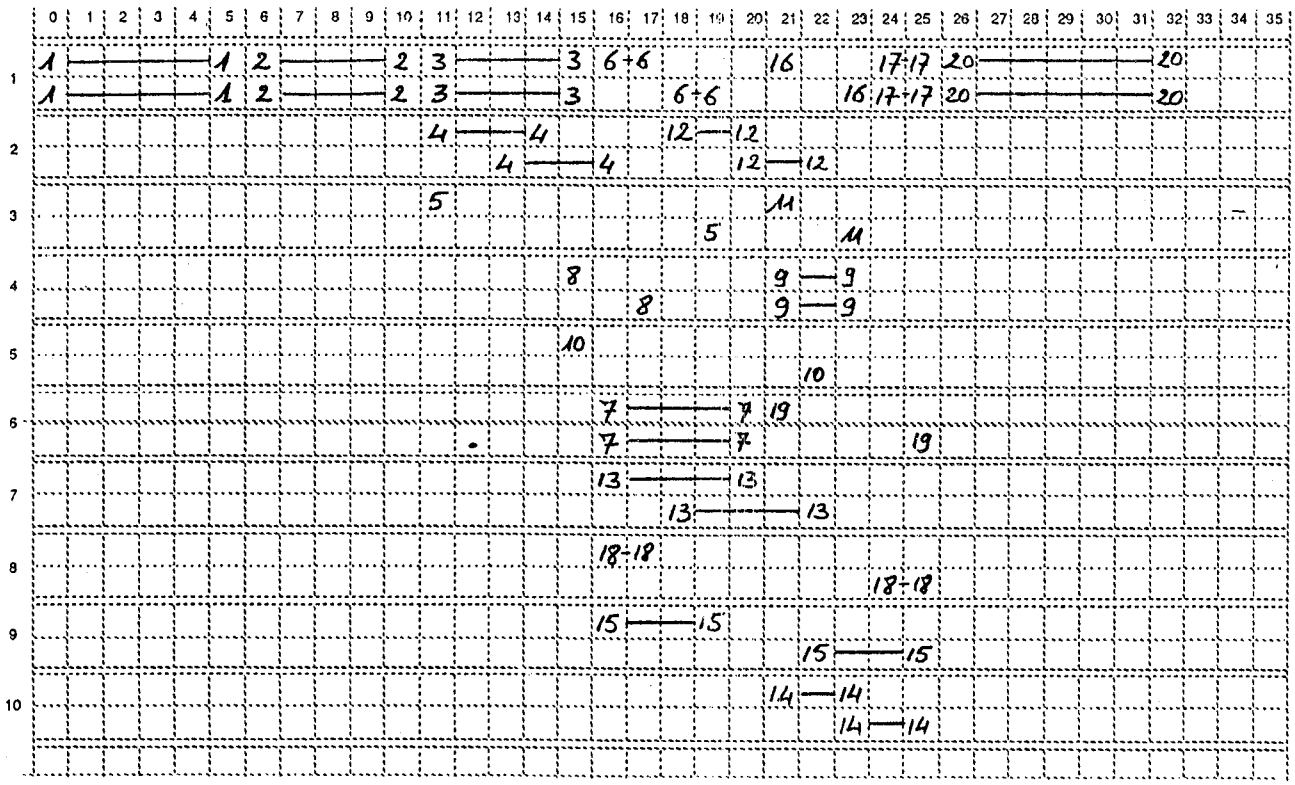
Représentation basée sur le réseau.





Représentation basée sur les résultats.





Ordonnancement avec ressources unitaires et méthode de recuit.

1.Présentation

Dans ce chapitre, nous nous intéresserons à des problèmes d'ordonnancement
- à liaisons de type Fin-Début ou Début-Début à valeurs positives ou nulles,
- sans contrainte négative de blocage,
- à contraintes potentielles,

dont nous cherchons à minimiser le délai total d'exécution. Mais nous supposons de plus que nous disposons de moyens ou ressources (hommes, machines, etc...) en quantité limitée. Chaque tâche, pour s'éxécuter requiert la mise à disposition d'une partie de ces moyens.

On notera $\mathcal{G}$= (X,E) le graphe d'ordonnancement à contraintes potentielles. Les tâches seront numérotées de 1 à N, (1 = sommet début, N = sommet fin). Les ressources seront numérotées de 1 à P. La ressource k sera en quantité Qk et la tâche i en demandera la quantité Qik (Qik $\leq$Qk, $\forall$i). On dira que le problème est à ressource unitaire si Qk=1 pour tout k.

La présence de ressources conduit à introduire deux types de contraintes nouvelles:

- des contraintes disjonctives entre deux tâches i et j, exprimant le fait que i et j doivent s'exécuter dans des intervalles de temps disjoints. Ces contraintes se trouvent lorsque i et j utilisent une ressource k et que l'on a: Qik+Qjk > Qk.

- des contraintes cumulatives exprimant le fait que les moyens sont limités. Ces contraintes se rencontrent lorsqu'à un instant t donné, un ensemble $i_1, i_2..,i_p$ de tâches utilisant une même ressource sont susceptibles d'être exécutées et que l'on a :

$\sum Q i_l k > Qk$.

Complexité.

Le problème posé est NP-complet dès qu'il y a une ressource, même si elle est en quantité unitaire. L'ordonnancement sur © = (X,E) détermine des dates de début au plus tôt dtot(x) et de fin au plus tard ftard(x). Pour prendre en compte la ressource, il faut fixer un ensemble de date de début d'exécution t(x) tel que:

1/ deux tâches ne soient pas exécutées simultanément:

$$\forall (x,y) \in X*X,\ t(y) \geq t(x)+duree(x) \text{ ou } t(x) \geq t(y)+duree(y)$$

2/ x ne commence pas avant dtot(x)

$$\forall x \in X,\ t(x) \geq dtot(x)$$

3/ x s'achève avant ftard(x)

$$\forall x \in X,\ t(x)+duree(x) \leq ftard(x)$$

S'il n'existe pas d'ordonnancement admissible, on abandonne la condition 3 et on cherche un ordonnancement de retard minimal = Max (0, t(x)-ftard(x)); le retard de l'ordonnancement est alors le retard maximum.

Ainsi formulé, ce problème est identique au problème dit "à une machine" (le problème général à m machines se pose par exemple dans un centre informatique disposant de m processeurs (spécialisés ou non); un ensemble de travaux liés par des contraintes doivent être exécutés quotidiennement et l'on cherche un ordonnancement satisfaisant certains critères. Celui-ci est réductible au problème du partitionnement d'un ensemble E d'entiers en deux parties E1 et E2 de somme égale. Ce problème étant NP-complet, il en est de même du problème à une machine (Lenstra [LE]) et du problème de l'ordonnancement avec ressources.

2. Problème à ressource unitaire.

On se limite ici au cas où Qk=1 ∀k et Qik ∈ {0, 1}. Bien que cette restriction soit forte, ce problème recouvre de nombreux cas concrets. Il englobe non seulement certains problèmes de planification de projet proprement dit , le problème "à une

machine" avec dates au plus tôt et plus tard, mais aussi des problèmes " d'atelier". Le problème d'atelier apparait en contexte industriel. Un atelier contient m machines différentes. Il doit produire n pièces et chacune des n pièces doit subir diverses opérations d'usinage. On cherche à déterminer une gamme d'usinage minimisant la durée de travail. On parle de problème préemptif si le passage d'une pièce sur une machine peut être morcelé, non préemptif dans le cas contraire. Dans la littérature, on distingue différentes catégories de problèmes selon les contraintes imposées aux tâches:

(1) Flow shop

Une opération d'usinage nécessite de passer la pièce sur une seule machine. L'ordre de passage des pièces sur les machines est imposé et reste le même pour chaque pièce.

(2) Job shop

L'ordre de passage reste imposé, mais peut différer selon les pièces.

(3) Open shop

Aucun ordre n'est imposé.

Ici, nous considérerons de plus des problèmes d'atelier plus larges:

(4) Flexibilité sur l'ordre d'usinage d'une pièce

Les opérations d'usinage suivent une relation de préordre.

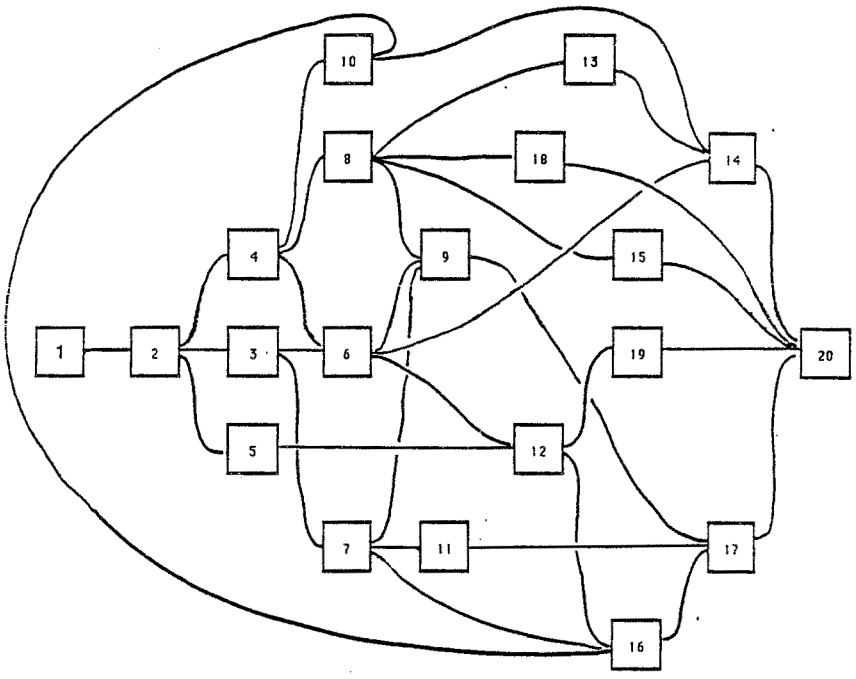
Dans le paragraphe suivant, nous montrerons comment modéliser ces divers problèmes pour les ramener aux problèmes d'ordonnancement à ressources unitaires.

Modélisation de cas type

1. **Planification** de projet lorsque le personnel disponible est scindé en équipes (réduites éventuellement à un seul individu) et qu'une tâche requiert l'emploi d'une partie de ces équipes (cas trivial):

- le graphe est le graphe du projet.
- une ressource est une équipe.

Exemple 1: on cherche à minimiser la durée du projet dont le graphe est donné ci-après. On dispose de 4 équipes pour réaliser le travail (cf tableau).



tâche	durée	équipe 1	équipe 2	équipe 3	équipe 4
1	0	0	0	0	0
2	2	1	1	0	0
3	2	1	0	0	0
4	4	0	1	0	1
5	9	0	0	1	1
6	8	1	0	1	0
7	1	1	1	1	1
8	9	0	1	0	0
9	6	1	0	0	0
10	9	1	1	1	1
11	9	0	1	1	0
12	5	1	0	0	1
13	2	0	0	0	1
14	6	1	1	1	1
15	3	1	1	0	1
16	9	0	1	0	1
17	7	1	0	0	0
18	8	1	1	1	1
19	10	1	1	0	1
20	0	0	0	0	0

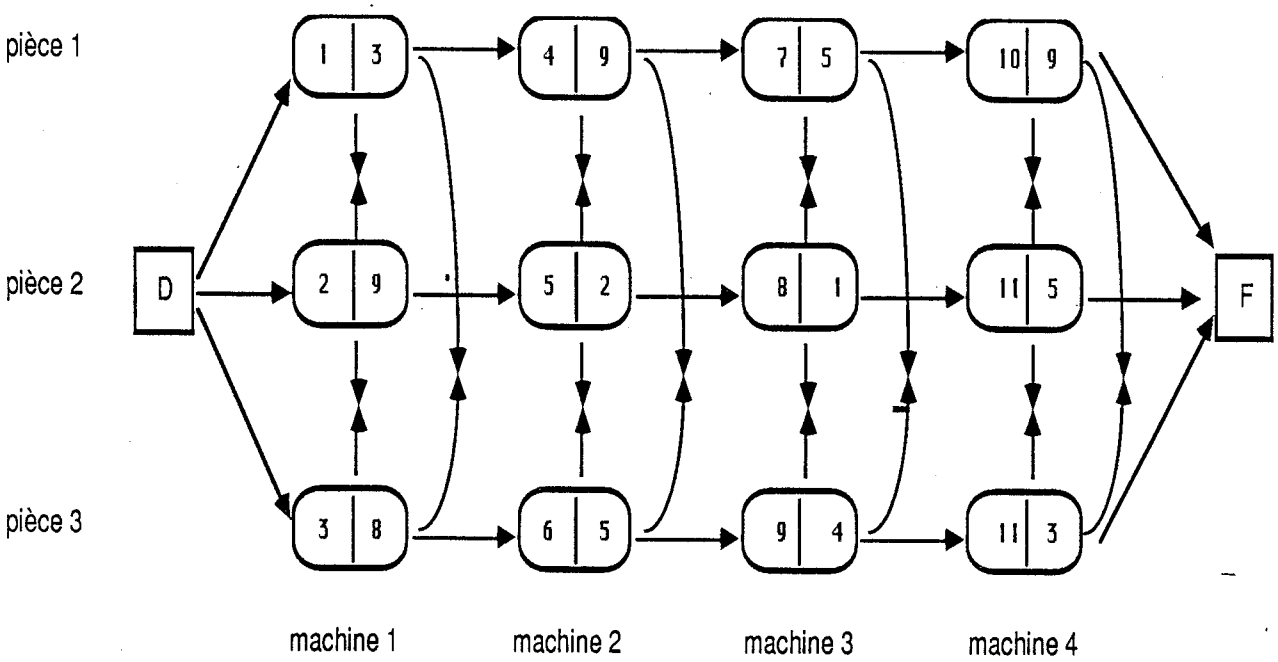
2. **"Flow Shop", "job shop" ou "open shop"** non préemptifs:

- une tâche correspond au passage d'une pièce sur une machine. Par commodité nous noterons Pi-Mk la tâche correspondant à la pièce i (i=1..n) sur la machine k (k=1..p).

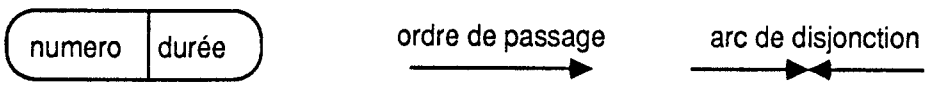
- on crée un arc entre Pi-Mk_1 et Pi-Mk_2 si la pièce i passe sur la machine k_2 immédiatement après son passage sur k_1. La longueur de l'arc est la durée du temps de passage sur k_1.

- les ressources sont les machines et Pi-Mk utilise la ressource k. Ceci assure que les pièces passent à tour de rôle sur la machine.

exemple 2: 'flow shop' tiré de Carlier [CA]



légende

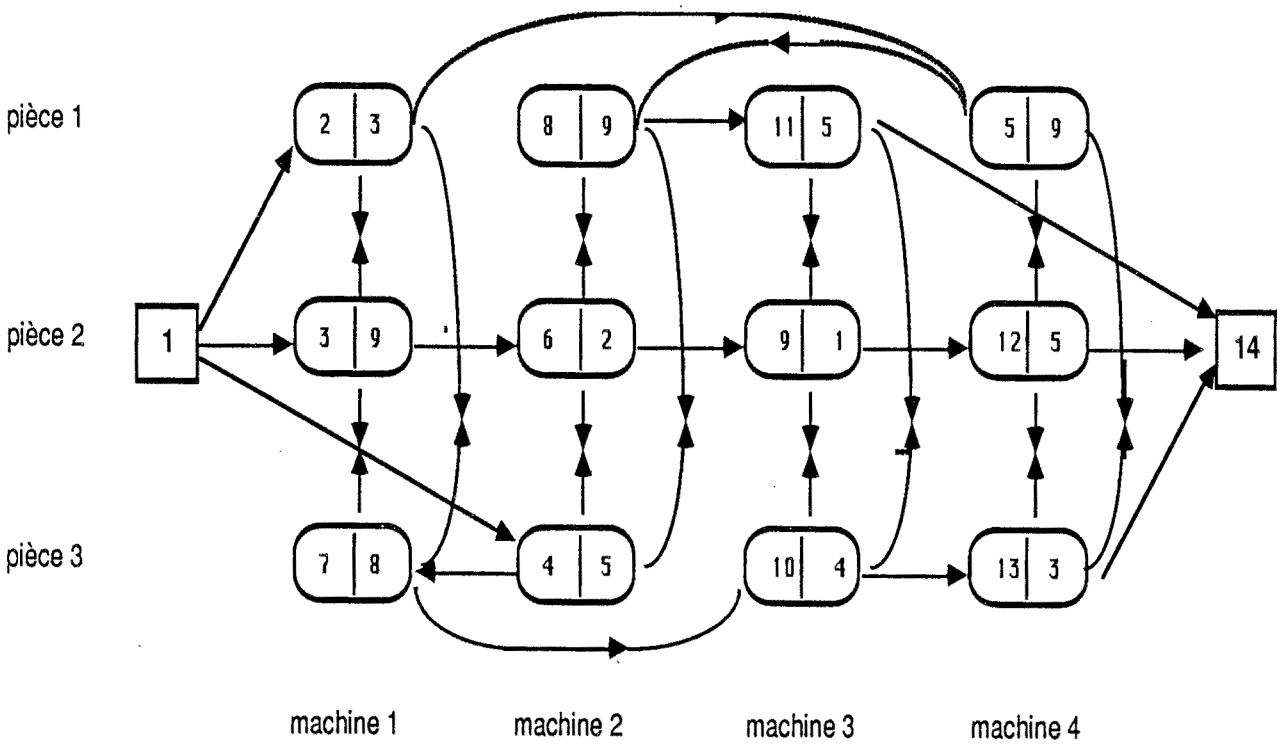


Exemple 3: 'job shop', il y a toujours 3 pièces sur 4 machines, mais les ordres de passages sont:

pour la pièce 1: 1, 4, 2, 3

pour la pièce 2: 1, 2, 3, 4

pour la pièce 3: 2, 1, 3, 4



3. Problème non préemptif avec **flexibilité** sur l'ordre d'usinage des pièces.

- les tâches sont identiques à celles du flow shop.

- on crée un arc entre Pi-Mk_1 et Pi-Mk_2 si la pièce i doit être passée sur la machine k_1 avant de passer sur k_2. La longueur de l'arc est la durée du temps de passage sur k_1.

- les ressources sont les machines et les pièces. La tâche Pi-Mk utilise la ressource machine k, comme ci dessus, et la ressource pièce i. Ceci assure que la pièce i subisse toutes ses opérations d'usinages.

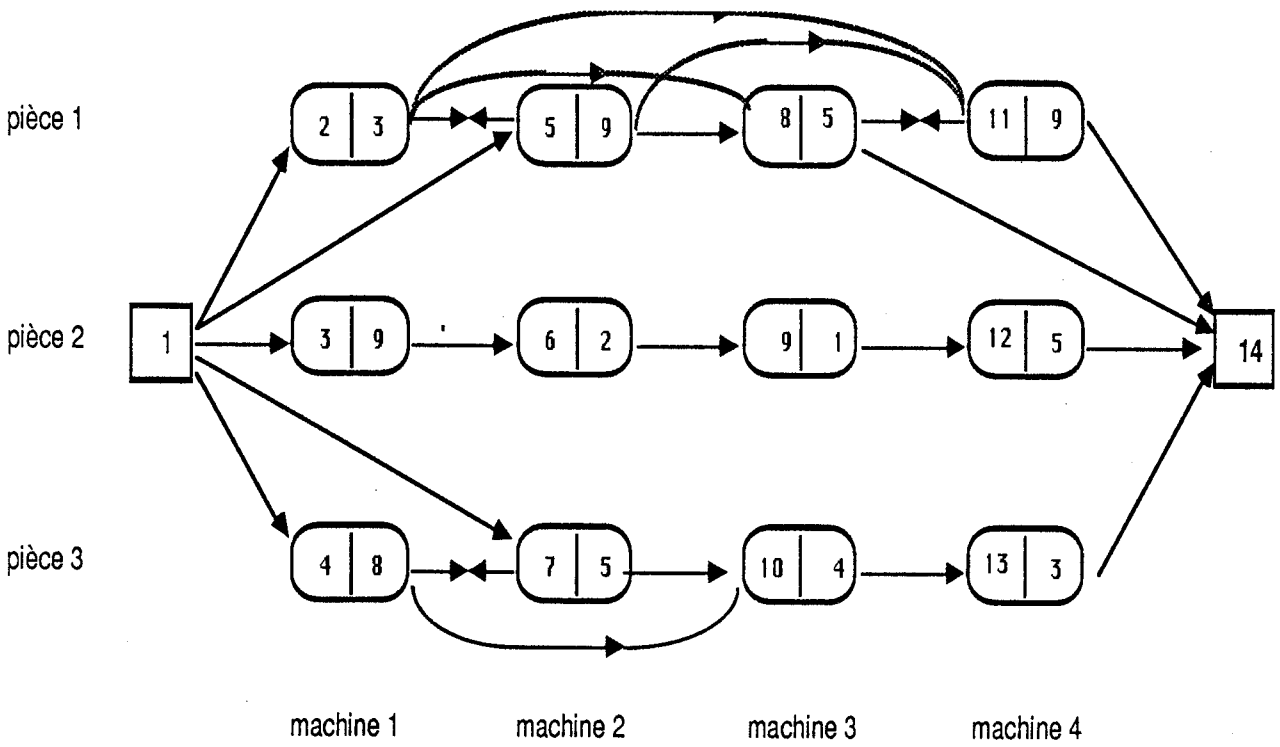
Exemple 4: l'ordre de passage est ici:

pour la pièce 1: 1 ou 2, puis 3 ou 4.

pour la pièce 2: 1, 2, 3, 4

pour la pièce 3 : 1 ou 2, puis 3 et 4.

Remarque: sur la figure, ne sont représentés que les arcs de disjonctions correspondant aux pièces.



Formulation.

Le problème à ressource unitaire, crée uniquement des contraintes disjonctives. Formellement, une contrainte disjonctive introduit deux arcs disjonctifs (i,j) et (j,i) valués par Pi et Pj. Arbitrer une contrainte disjonctive, c'est choisir l'ordre d'exécution des deux tâches, ce qui équivaut à ajouter au graphe $\mathcal{G}$, soit (i,j) soit (j,i). Soit $Xk \in X$, l'ensemble des tâches utilisant la ressource k. Les éléments de Xk sont en disjonction deux à deux et le graphe ainsi formé est complet. On dira que Xk forme une clique de disjonction.

Un arbitrage $\mathcal{A}$ est un ensemble d'arcs disjonctifs. L'appartenance de (i,j) à $\mathcal{A}$ impose d'éxécuter i avant la tâche j. Un arbitrage est dit complet si toutes les contraintes disjonctives sont arbitrées. Un arbitage est dit compatible, s'il n'introduit pas de circuit absorbant. Les graphes que nous traitons ici, sont des graphes sans circuit; par suite un arbitrage sera dit compatible si le graphe $(X, E \cup \mathcal{A})$ n'a pas de circuit. On appellera solution, un arbitrage S, complet et compatible. On rappelle qu'un tri topologique d'un graphe transitif G=(V,E) est une renumérotation t(x) des sommets x de X, telle que:

- $1 \leq t(x) \leq |V|$
- $\forall xy \in X^*X,\ t(x) \neq t(y)$
- $\forall (xy) \in E,\ t(x) < t(y)$.

Par la suite, on appellera tri toute suite ordonnée $T = \{ x_1, x_2, \ldots, x_n \}$ des sommets de $\mathcal{G}$, telle que la renumérotation $t(x_i) = i$ donne un tri topologique. On notera " < " la relation d'ordre induite par T sur les tâches.

lemme 1: soit T un tri de $G = (X, E \cup S)$: alors T est un tri de $\mathcal{G}=(X,E)$.

En effet: $(xy) \in E \Rightarrow (xy) \in E \cup S \Rightarrow t(x) < t(y)$.

lemme 2: soit T un tri de $\mathcal{G}=(X,E)$; considérons l'arbitrage $\mathcal{A}(T)$ défini sur T de la manière suivante: s'il existe une contrainte disjonctive xy, on garde l'arc xy si x<y,

l'arc yx si y<x. $\mathcal{A}(T)$ constitue une solution.

- l'arbitrage est complet: puisque T est une relation d'ordre total sur X, toute containte xy est arbitrée de manière unique,

- l'arbitrage est compatible soit un arc xy de $G = (X, E \cup \mathcal{A}(T))$; si $xy \in E$ alors x<y par définition de T; si $xy \in \mathcal{A}(T)$ alors x<y par construction de $\mathcal{A}(T)$; par suite l'existence d'un circuit passant par les deux sommets s1 et s2 impliquerait que s1<s2 et s2<s1.

théorème: $\mathcal{A}$ défini une surjection de l'ensemble des tris topologiques de $\mathcal{O}$ dans l'ensemble des solutions.

- tout tri T définit une solution

- toute solution S s'obtient à partir des tris communs à $\mathcal{O}=(X,E)$ et $G=(X,E \cup S)$.

Par suite, il est possible d'explorer l'ensemble des solutions par exploration des tris topologiques du graphe d'ordonnancement de départ.

Passage entre tris

lemme 1: soit T= un tri sur le graphe transitif G=(V,E) et x,y deux tâches consécutives sur T. Si $xy \notin E$, alors en permutant x et y, on obtient un nouveau tri $\mathcal{T}$. On parlera de permutation élémentaire.

lemme 2: soit T et S, deux tris sur G. Il est possible de passer de T à S par une suite finie de permutations élémentaires.

Soit x le sommet de S le plus à gauche et différenciant S de T. T et S sont de la forme: $T = \{ z_1, z_2, \ldots z_{p-1}, y_1, y_2, \ldots y_{q-1}, x, \ldots \}$

$S = \{ z_1, z_2, \ldots z_{p-1}, x, \ldots \}$.

Par définition de S, tous les prédécesseurs de x sont dans $\{ z_1 \ldots z_{p-1} \}$. Par suite, il est possible dans T de permuter successivement y_{q-1} et x, y_{q-2} et x ... y_1 et x. On pose $z_p = x$ et l'on répète l'opération.

Permutation de deux sommets consécutifs.

Par la suite, connaissant la date de fin de projet pour un tri donné, il serait intéressant de pouvoir déterminer la nouvelle fin du projet après une permutation élémentaire, sans recalculer l'ordonnancement.

Soit T un ordre donné, $\mathcal{A}(T)$ l'arbitrage correspondant; on connait les dates ftard(x) sur G=(X, E ∪ A). Considérons T* l'ordre obtenu par la permutation élémentaire entre y et x (par définition y n'est pas un prédecesseur de x). On posera Cx la date de fin au plus tôt de x après permutation et Rx= Cx - ftard(x).

1. $(yx) \notin \mathcal{A}(T)$: alors $\mathcal{A}(T) = \mathcal{A}(T^*)$ et la permutation est sans effet. (cas 1)

2. $(yx) \in \mathcal{A}(T)$ et ftot(y) < dtot(x):

   alors ∃ z tel que z soit bloquant. Par suite dtot(x)= ftot(z).

   Après permutation, z est toujours bloquant:

   Cx = ftot(z)+duree(x) = dtot(x)+ duree(x)

   et Cy = Cx+durée(y) > ftot(y)

   Si Cy ≤ ftard(y) la permutation n'entraine pas de retard global (cas 2)

   Si Cy > ftard(y), la permutation entraine un retard global de Cy- ftard(y) (cas 3)

3. $(yx) \in \mathcal{A}(T)$ et ftot(y) = dtot(x):

   y est bloquant pour x; aprés permutation, on aura:

   Cx ≤ ftot(x)

   Cy = Max (ftot(y), Cx + durée(y))

   si Cy ≤ ftard(y), alors

   - si x était critique, le chemin critique passant par x est réduit de ftard(x)-Cx. la durée globale du projet sera réduite d'une valeur comprise entre 0 et ftard(x)-Cx. (cas 4)
   - si x était non critique, la date de fin de projet sera inchangée (cas 5).

si Cy > ftard(y)

si Cy ≤ ftard(x), le projet prendra un retard compris entre 0 et Cy- ftard(y),

selon que Succ(x) ∩ Succ(y) est vide ou non (cas 6)

si Cy > ftard(x) le projet prendra un retard compris entre

Min (Cy- ftard(y), Cy- ftard(x)) et Max (Cy- ftard(y),Cy- ftard(x)) (cas 7)

En résumé, 4 situations peuvent se présenter:

1: la permutation ne change pas la durée du projet cas 1,2,5

2: la permutation peut supprimer un chemin critique cas 4

3: la permutation peut entrainer un retard du projet cas 6

4: la permutation entraine un retard du projet cas 3,7.

On constate qu'il n'est pas possible par simple examen de x et de y, de connaitre précisément les répercussions chiffrées d'une permutation élémentaire et qu'en pratique, le recalcul partiel de l'ordonnancement s'impose.

3. Méthode de Monte Carlo et méthode de recuit simulé.

La méthode exhaustive pour déterminer l'optimum, consisterait à générer tous les tris et à estimer les solutions correspondantes. Générer tous les tris d'un graphe est un problème algorithmiquement facile: si on suppose qu'au pas i, on connaisse les i premiers sommets du tri, soit $Ti = \{ x_1 , x_2 ,..., x_i \}$, sont candidats à entrer au pas i+1, les sommets y tels que:

- $y \notin Ti$
- $\forall z \in Pred(y), z \in Ti.$

Ceci peut constituer un intéressant exercice sur la récursivité. On associe aux sommets le vecteur logique M tel que M(i)=vrai si $i \in Ti$ et le vecteur O(i)=X qui donne le sommet X à la ième position du tri. On définit le sous traitement Range(e: X, I) qui range le sommet X en ième position de la manière suivante:

```
algorithme Range  (e: X,I : entiers)
Début
     O(I) := X
     M(X) := vrai
   – si I = N alors écrire le tri O(1) ... O(N)
     sinon
          pour y de 2 à N faire
               si non M(y) et M(z) ∀ z ∈ Pred(y) alors Range(y,I+1)
          fpour
     finsi
     M(X) := faux
Fin
```

L'ensemble des tris est généré par les instructions:

```
     pour i de 1 à N faire M(i) := faux
     Range (1,1)
```

Malgré la vitesse d'éxécution des machines actuelles, cette méthode n'est pas raisonnable en temps. Pour fixer les idées, le graphe de l'exemple 1 (20 sommets) a plus de 100 000 000 de tris. Cependant, bien que l'on ne dispose pas d'un temps de calcul illimité, le micro-ordinateur permet d'envisager des algorithmes "gourmands". Dans la mesure où l'utilisation du micro ne coûte (presque) rien, il est facile de laisser tourner le programme toute une nuit, par exemple. Ceci donne un regain d'interêt aux méthodes basées sur des tirages aléatoires de solutions.

Le principe général des méthodes dites de Monte-Carlo est de générer aléatoirement une suite de P solutions, d'estimer la fonction objective de chacune et de garder la meilleure.

Plus récemment, Kirkpatrick ([KI] 1982) reprenant une idée développée par Metropolis ([ME] 1953), proposait une méthode dite de "recuit", pour résoudre les problèmes d'optimisation complexes. On cherche ici, à simuler la méthode de recuit utilisée par les métallurgistes pour éliminer les défauts cristallins dans un métal. Pour cela, on réchauffe le métal et on le laisse se refroidir lentement. Pour le physicien, ce procédé a pour effet d'amener le métal dans un état d'énergie minimum. Le point à souligner, est qu'à l'état naturel, l'énergie ne diminue pas de manière continue. Vu le nombre des atomes mis en jeu (10^{23} par cm^3), seul le comportement moyen du

système à une température donnée, peut être observé, le système fluctuant légèrement autour de ce comportement. Il y a donc une certaine probabilité de constater dans le temps une augmentation d'énergie, et cette probabilité donnée par l'équation de Boltzmann , est de la forme:

$$P(\delta E) = \exp (- \delta E / k_B T)$$

où T est la température et k_B la constante de Boltzmann.

Considérons maintenant un problème d'optimisation dont on cherche à minimiser une fonction objectif f. La méthode très générale de recuit est alors:

- fixer un paramètre T à une valeur élévée (la température)
- définir une configuration de départ Ca
- répéter
  pour T donné, répéter un certain nombre de fois:
  générer aléatoirement une configuration voisine Cv de la configuration actuelle Ca. Poser $\delta E = f(Cv) - f(Ca)$
  accepter Cv comme nouvelle configuration actuelle avec la probabilité:
  $P=1$ si $\delta E < 0$
  $P = \exp (- \delta E / kT)$ si $\delta E > 0$
  fin
  baisser T
  jusqu'à T proche de 0.

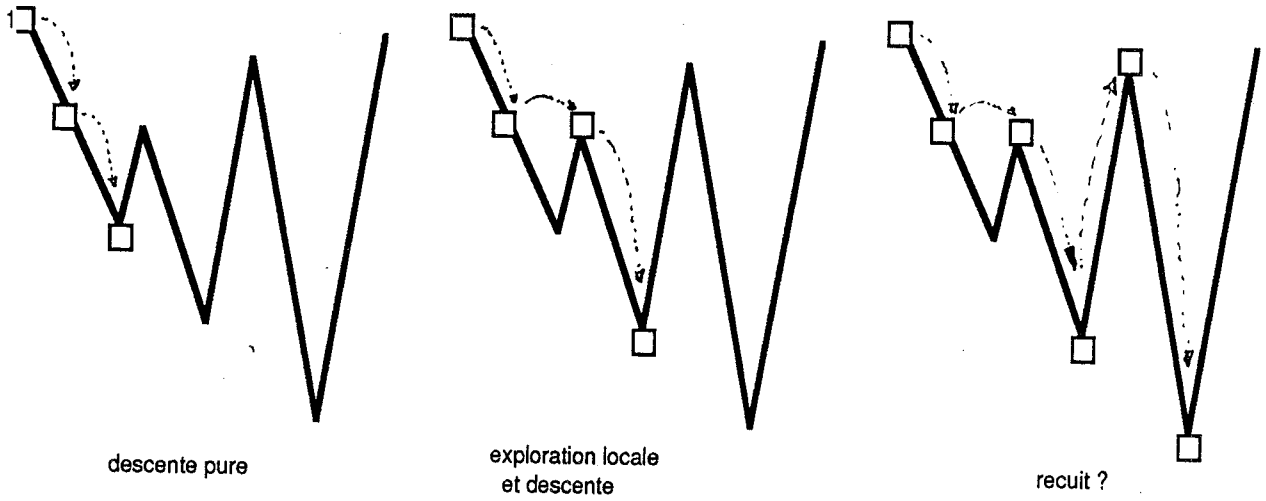
Le terme configuration s'entend:

- soit comme solution réalisable (et f est la fonction objectif du problème)
- soit comme solution au sens large, f étant alors la fonction objectif du problème, augmentée d'un facteur mesurant l'irréalisibilité.

Monte Carlo équivaut à un recuit à température infinie. Si l'on fixe dès le début la température à 0 (recuit à température nulle), on obtient un algorithme du type exploration locale et descente.

Dans le cas de problèmes dont l'ensemble des solutions présente de nombreuses vallées (les problèmes NP-complet notamment), la difficulté essentielle est de localiser la vallée la plus basse. Les méthodes de descente pure, aboutissent fatalement au fond de la vallée de départ; donc en général à un minimum local. Les méthodes

travaillant par exploration locale et retenant les solutions améliorantes, permettent elles, de franchir les petits cols. L'originalité du recuit, est qu'il permet d'augmenter la taille des cols que l'on franchit et de passer d'une vallée à une autre.



Comme pour toute heuristique, l'optimalité de la solution trouvée en fin d'algorithme, n'est pas garantie. Dans le recuit, il peut arriver qu'elle soit plus mauvaise que certaines solutions trouvées en cours d'exploration. En pratique, on gardera donc la meilleure solution explorée et on la prendra comme solution en fin de calcul.

Le recuit nécessite, l'exploration d'un grand nombre de configurations. Lorsque la méthode donne de bons résultats, la question est souvent posée de savoir si ces résultats:

sont dus à la méthode;

ou sont dus au grand nombre de configurations explorées.

Supposons en effet qu'une configuration sur 1000 donne une valeur optimale. Le sens commun est porté à estimer que toute méthode explorant plus de 1000 configurations trouvera "à coup sur" l'optimum. Afin de quantifier cette impression, faisons un bref rappel de probabilité. Explorer n configurations sachant que la probabilité qu'une configuration soit optimale est p, est équivalent à faire n tirages d'une boule, avec remise, dans un sac contenant des boules blanches et noires en

proportion p et q=1-p. L'evênement "obtenir au moins une boule blanche en n tirages" à pour complémentaire "obtenir une noire à chaque tirage". Par suite:

$$P(\text{au moins 1 blanche}) = 1 - q^n.$$

En fait, le sens commun n'est pas très bon juge. La probabilité d'obtenir l'optimum en 1000 coups n'est que de 63% et il faudrait 4600 tirages pour arriver à 99%.

4. APPLICATIONS.

4.1 Génération aléatoire d'une solution.

Les principes de base sont identiques à ceux utilisés pour générer tous les tris: si l'on connait les i premiers sommets du tri, soit $Ti = \{ x_1, x_2, \ldots, x_i \}$, sont candidats à entrer au pas i+1, les sommets y tels que:

- $y \notin Ti$
- $\forall z \in Pred(y), z \in Ti.$

L'algorithme très général pour générer une solution est:

```
debut
    numéroter 1 le sommet Début
    pour i de 2 à N
        déterminer l'ensemble C des sommets y candidats à entrer
        choisir un des sommets de C, soit y0
        numéroter i le sommet y0
    finpour
fin.
```

Le type de solution recherchée repose entièrement sur la manière de choisir un des sommets de C. Dans le cas de méthodes sérielles, on prendra par exemple la tâche de durée minimale ou celle induisant l'avancée minimale du projet. Pour obtenir une solution aléatoire, on tirera au hasard l'un des sommets de C.

Cette méthode est d'ordre $\mathcal{O}(n^2)$. Lorsqu'on veut obtenir un échantillon représentatif de l'ensemble des solutions, cela nécessite de générer un nombre r de solutions et l'ordre devient $\mathcal{O}(r.n^2)$. On a établi précedemment que l'on passe d'un tri

à un autre par une suite finie de permutations élémentaires. Si , partant d'un tri, on génère un nouveau tri en tirant au hasard une longue suite de permutations, il semble raisonnable d'estimer que ce nouveau tri est peu dépendant du tri initial; ou plus précisément, qu'un grand nombre de tris ainsi générés, forment eux-aussi un échantillon représentatif. En se fixant comme longueur de la suite, un multiple de n, cette méthode est en ©(r.n). Un test effectué sur un réseau à 100 sommets et consistant à générer 2000 tris par la première méthode et 2000 tris en tirant 2*n permutations s'est révélé positif.

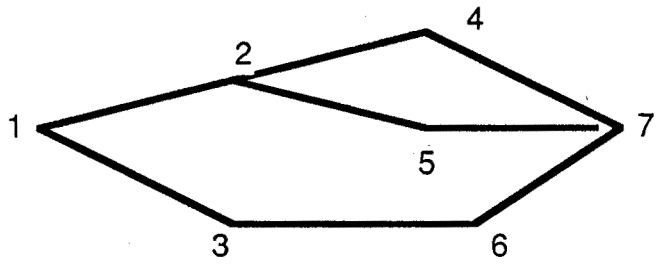
4.2 Méthode de recuit simulé.

Détermination des configurations voisines.

Dans le cas présent, une configuration sera un tri, induisant donc une solution réalisable et la fonction objective sera la minimisation du temps total du projet. La principale difficulté de cette méthode, est de définir une bonne relation de voisinage entre configurations. Dans les problèmes où le recuit a déjà été testé (problème de tournée par exemple [BO]), on reprend les configurations générées par la méthode du 2-opt. Dans notre cas, cela reviendrait à choisir 2 sommets et à les permuter si cela est possible. Appellons X et Y ces deux sommets (avec X < Y dans le tri initial). Si X et Y ne sont pas consécutifs, il faut:

- tout d'abord déterminer s'il n'existe pas de chemin entre X et Y;
- lorsque la permutation est possible, permuter X et Y;
- placer après X les sommets Z initialement compris entre X et Y, et dont X est un antécédent
- placer avant Y les sommets W initialement compris entre X et Y, et dont Yest un descendant

L'ensemble de ces opérations aboutit à un tri éloigné de la simple permutation de deux sommets. A titre d'illustration, si l'on veut permuter 2 et 6 dans ce graphe:



on passe de: 1, 2, 3, 4, 5, 6, 7

à: 1, 3, 6, 2, 4, 5, 7.

Si on choisi X et Y consécutifs, on effectue une permutation élémentaire. La solution obtenue est améliorante uniquement dans le cas 4 et si le chemin critique est unique. Ceci explique, qu'expérimentalement, il faille explorer un grand nombre de configurations pour obtenir des résultats probants. Pour les petits graphes, l'efficacité est équivalente à celle de Monte Carlo. Par exemple, si l'on explore 2000 configurations dans l'exemple 1 (100 sauts de température et 20 essais par saut), le recuit trouve l'optimum dans 95% des tests. Or, de son coté, la méthode de Monte-Carlo, en explorant 2000 configurations aléatoires, aboutit aussi à l'optimum dans plus de 95% des cas. Afin de limiter le nombre de configurations explorées, la solution retenue finalement a été de prendre comme configuration voisine, le tri obtenu après un petit nombre de permutations élémentaires tirées au hasard. Expérimentalement, il apparait que tirer de 1 à 10 sommets et effectuer la permutation élémentaire correspondante, si elle est possible, donne de bons résultats pour les graphes considérés.

Schéma de refroidissement.

En physique statistique, on sait que la température doit décroitre par palier, mais la stratégie de décroissance reste très empirique. Dans la stratégie retenue ici, on se fixe le nombre de paliers n, et le nombre d'essais par palier e; ceci pour connaître précisément le nombre de configurations explorées. La décroissance choisie est une décroissance géomètrique. En notant Tn, Tn-1,... T1 les températures (avec Tn température maximale de départ et T1 température d'arrêt), on a donc:

$Ti := \alpha.Ti+1 := T1 . \alpha^i \quad (\alpha < 1).$

Afin de rendre l'algorithme indépendant de la durée du projet, on a adopté les règles suivantes:

- à la température maximale, on accepte avec une probabilité forte β_n une dégradation importante ∂_n de la solution initiale f°.

- à la température d'arrêt, on accepte avec une probabilité faible β_1 une dégradation minime δ_1 (ici 1 puisque les solutions sont entières), ce qui revient à définir la notion de gel de la solution.

On a donc:

$1 / k = T1 * \log (\beta_1)$

$Tmax = f° * \partial_n * 1/k / \log(\beta_n)$

$\alpha = \exp (\log(Tn/T1) / n).$

Une configuration Cv sera acceptée à la température Ti avec la probabilité P:

P = 1 si f(Cv) - f(Ca) < 0

P = exp ((f(Cv) - f(Ca)) / k / Ti)

= exp ((f(Cv) - f(Ca) * log (β_1) / α^i) sinon.

Ces équations sont paramétrées par T1. Dans les tests, on a pris:

T1 = 1, β_n = 80%, δ_n = 10%, β_1 = 1% et n=100.

Soit avec f° = 100: Tmax = 206 α = 1,055

avec f° = 500: Tmax = 1030 α = 1,071

5. Expérimentation 1.

Les programmes ont été écrits en Pascal TML sur MacIntosh. Le premier test a porté sur le problème de l'exemple 1. Afin d'avoir une idée de l'ensemble des solutions, on a généré aléatoirement (méthode 1) 50000 solutions, ce qui a demandé 1 heure. La figure suivante donne les valeurs trouvées, le nombre d'occurence et la fréquence d'apparition de chaque valeur en 1/10000.

85	64	12,80	
86	59	11,80	
87	284	56,80	
88	212	42,40	
89	461	92,20	
90	1101	220,20	
91	665	133,00	
92	2953	590,60	
93	2056	411,20	
94	1840	368,00	
95	376	75,20	
96	2588	517,60	
97	1055	211,00	
98	4075	815,00	
99	4716	943,20	
100	2282	456,40	
101	6341	1268,20	
102	1139	227,80	
103	2095	419,00	
104	343	68,60	
105	4698	939,60	
106	0	0,00	
107	8149	1629,80	
108	0	0,00	
109	2478	495,60	

Remarque: sur cet exemple, il est possible de montrer que la valeur 85 est l'optimum (cf annexe 1).

1. la fréquence d'apparition du groupe en1/10000;
2.le nombre théorique de tirages nécessaires pour l'obtenir dans 95% des cas;
3. le nombre théorique de tirages nécessaires pour l'obtenir dans 99% des cas.

groupe	fréquence	tirages à 95 %	tirages à 99 %
85	12.8	1575	2421
85 ou 86	24.6	830	1277
85 à 87	81.4	355	546

La méthode de recuit a été appliquée 100 fois, avec 100 sauts de température et tirage de 10 sommets, avec une configuration de départ fixe (solution de valeur 100), et en faisant 1, 2 puis 3 essais par température (soit 100, 200, 300 configurations explorées). Afin de comparer recuit et Monte-Carlo, on a calculé à partir des données de l'échantillon, le nombre de fois où les valeurs 85, 86 et 87 auraient été solution en appliquant 100 fois Monte-Carlo sur un tirage de 100, 200 et 300 configurations. En dernier lieu, afin de savoir si le fait d'accepter certaines dégradations a une incidence réelle sur les résultats, des calculs, similaires au recuit, ont été faits en n'acceptant que les configurations améliorantes (recuit à température nulle). Les résultats sont donnés par les 3 tableaux suivants.

1 essai	Valeur	Recuit	Monte Carlo	Ameliorant
	85	78	17	70
	86	12	13	11
	87	8	27	15
	88	2	----	2
	90	0	----	2

2 essais	Valeur	Recuit	Monte Carlo	Ameliorant
	85	94	32	85
	86	4	19	2
	87	2	31	12
	88	0	----	1

3 essais	Valeur	Recuit	Monte Carlo	Ameliorant
	85	99	43	94
	86	1	23	1
	87	0	22	5

Sur cet exemple, le recuit apparait nettement plus performant que la méthode de Monte Carlo: le recuit trouve l'optimum dans 99% des cas, en explorant 300 configurations, alors que Monte carlo en demanderait l'examen de 2400. Au vu des tableaux, l'acceptation de dégradations locales de la solution a une influence réelle, mais semble moins jouer lorsque le nombre de configurations explorées augmente.

Afin de vérifier que les résultats satisfaisants obtenus ne sont pas dus à un choix "heureux" de la solution initiale, la même expérimentation a été conduite en générant au hasard la solution initiale. Pour 2 essais par température, on a procédé à 500 tests en fixant ou non la solution de départ afin d'avoir des données plus précises. On obtient:

2 essais	valeur	sol départ fixe	sol départ aléatoire
	85	469	447
	86	18	24
	87	10	19
	88	3	5
	89	0	1
	90	0	4

Si l'on prend comme hypothèse qu'il y a une probabilité 469/500 d'obtenir la valeur 85, et ce indépendemment de la solution initiale, on aurait du trouver 85 dans la seconde expérience dans 469 cas avec un écart type de 5,4 (d'après la loi de Bernouilli). Donc ici, le choix de la solution initiale a joué.

Avec 3 essais par température, l'influence de la solution initiale diminue (85 est trouvé dans 487 cas pour 493±5,15 cas attendus).

3 essais	valeur	sol départ fixe	sol départ aléatoire
	85	493	487
	86	6	8
	87	0	4
	88	1	0
	89	0	0
	90	0	1

Au vu de cet exemple, dans ce type de problèmes, le recuit montre ses avantages et ses limites:

- ses avantages: il "plonge" rapidement vers les solutions de valeurs faibles;
- ses limites: il demande l'examen d'un nombre élevé de configurations (ici 300) si l'on veut des résultats fiables, quelque soit le départ (ce qui correspond tout à fait à la pratique des métalurgistes qui exigent un refroidissement lent du métal).

Expérimentation 2.

Le recuit est une méthode adaptée aux gros problèmes combinatoires. Aussi, les expérimentations suivantes ont été faites sur un projet plus conséquent avec 100 tâches et 4 ressources. Chaque tâche a de 1 à 10 antécédents. Sur un échantillon de 71000 solutions générées de manière aléatoire, les valeurs trouvées vont de 379 à 481. On reprend ici les 2 valeurs les plus faibles:

valeur	occurence	fréquence	fréquence cumulée
379	1	0.1408	0.1408
382	1	0.1408	0.2817

Le tableau suivant résume pour chacun des 2 groupes de valeurs: 379, 379 ou 382:
1. la probabilité de l' obtenir en 2500 tirages
2. le nombre théorique de tirages nécessaires pour l'obtenir à 95% des cas;
3. le nombre théorique de tirages nécessaires pour l'obtenir à 99% des cas.

groupe	proba en 2500 tirages	tirages à 95%	tirages à 99%
379	3,46 %	212695	326964
379 ou 382	6,80 %	106347	163481

La méthode de recuit avec 100 sauts de température, tirage de 10 sommets, et avec une configuration de départ fixe (solution de valeur 414) a été testée en faisant varier le nombre d'essais à chaque température (25,50,200,400); ce qui équivaut à explorer de 2500 à 40000 configurations par recuit. Le tableau de la page suivante synthétise ces tests.

Vu les temps de calcul sur micro, il n'a pas été possible d'effectuer un grand nombre de recuits pour obtenir des données statistiquement exploitables. Cependant, certaines conclusions peuvent se dégager:

1. le recuit est nettement plus performant que la méthode de Monte Carlo, y compris lorsque le nombre de configurations voisines explorées est faible;

2. le fait d'accepter des dégradations locales,est ici primordial. Dix tests effectués en n'acceptant que les configurations améliorantes (recuit à T=0) avec 25 essais par température ont donnés des résultats compris entre 387 et 402 (à comparer avec 355 et 380). Cinq tests avec 200 essais par température ont donné des valeurs entre 386 et 392 (à comparer avec 351 et 359);

3. lorsque le nombre d'essais par température augmente:

- dans un premier temps, en moyenne les solutions sont meilleures et paraissent être plus groupées (difficile à affirmer sur quelques tests);
- passé un certain seuil (400 essais ici), il n'y a plus d'amélioration.

essais / tempér.	25	50	200	400	800
config. vues	2500	5000	20000	40000	80000
temps / recuit	7 mn	15 mn	1 h	2 h	4 h
solution					
348					
349					
350					
351					
352					
353					
354					
355					
356					
357					
358					
359					
360					
361					
362					
363					
364					
365					
366					
367					
368					
369					
370					
371					
372					
373					
374					
375					
376					
377					
378					
379					
380					
nombre d'essais	50	24	10	9	
moyenne	367,06	362,96	354,70	354,33	

L'expérimentation suivante a consisté à ne tirer qu'un seul sommet par essai. Lorsque l'on passe de n essais et 10 sommets à 10n essais et 1 sommet, le temps moyen d'un recuit est multiplié par 4 ou 5 (l'ordonnancement partiel à recalculer est plus petit). Les résultats en moyenne sont moins bons, bien que la valeur la plus faible (348) ait été trouvé lors de ces tests.

Un dernier test, qui s'est avéré peu probant, a consisté à mixer les deux expérimentations, en faisant:

- pendant les 50 premiers sauts de température: 25 essais en permutant 10 sommets, dans le but d'avoir une descente rapide;
- pendant les 50 derniers, 250 essais en permutant 1 sommet, dans le but d'avoir des voisinages plus fins.

Le tableau de la page suivante résume ces tests.

7. Conclusions.

Vu les temps de calcul éxigés pour tester le recuit sur un projet, il n'a pas été matériellement possible de procéder à des expérimentations plus nombreuses; aussi, doit-on considérer cette étude comme un premier débroussaillage du sujet et ces conclusions ne peuvent-elles être que partielles.

Le recuit en ordonnancement permet d'obtenir dans des temps raisonnables, y compris sur micro-ordinateur, une bonne solution réalisable et peut être quasi-optimale. Cette heuristique paraît prometteuse et il serait intéressant de consacrer des moyens informatiques conséquents à des tests plus complets.

essais / tempér.	250 essais	500 essais	1000 essais	25 à 10 puis 250 à 1
config. vues	25000	50000	100000	25000
temps / recuit	30 mn	1 h	1h 55 mn	
valeurs				
348				
349				
350				
351				
352				
353				
354				
355				
356				
357				
358				
359				
360				
361				
362				
363				
364				
365				
366				
367				
368				
369				
370				
371				
372				
373				
374				
375				
376				
377				
378				
379				
380				
nombre d'essais	20	10	10	20
moyenne	363,15	357,10	357,90	364,30

Annexe

Expérimentation 1: vérification de l'optimalité des solutions de valeur 85.

Dans les problèmes NP-complets, vérifier qu'une solution donnée est optimale est un problème difficile. Dans notre cas, la méthode exposée ci après, a permis de montrer assez facilement que les solutions de valeur 85 du projet de la page 98, sont optimales.

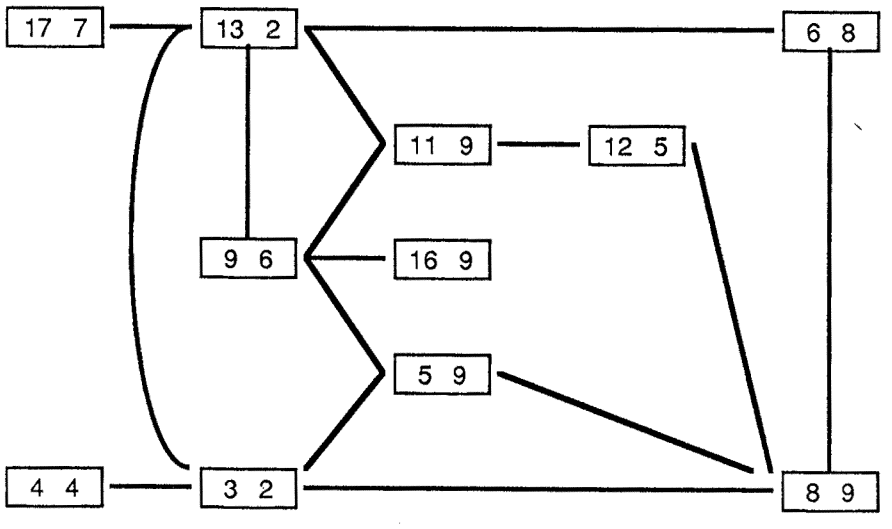
On appelera graphe de simultaneïté associé au projet le graphe valué S dont:

- l'ensemble des sommets est en bijection avec l'ensemble des tâches;
- la valeur d'un sommet est la durée de la tâche correspondante;
- il existe une arête (x,y) si et seulement si x et y peuvent être éxecutées simultanément; c'est à dire, s'il n'existe pas de contrainte d'antériorité liant x et y et si x et y n'emploient pas de ressource commune.

Un stable de S est donc un sous ensemble de tâches qui doivent être éxecutées dans des intervalles de temps distincts. Par suite, le poids d'un stable, est une borne inférieure de la durée optimale d_0 du projet. Déterminer une bonne borne inférieure du projet, revient donc à calculer la valeur s_0 du stable de poids maximum $\mathcal{S}$, problème lui-même NP-complet. Sur le projet à 20 tâches de l'expérimentation 1, le graphe de simultaneïté est le suivant:

- les sommets 2, 7, 10, 14, 15, 18, 19, 20 sont isolés (poids total 39);
- le sous graphe induit par les autres sommets et représenté ci-après, a pour stable de poids maximum { 4, 5, 6, 11, 16, 17 } (poids 46);

soit un poids total de 85.



Expérimentation 2.

L'ordonnancement utilisé pour la seconde expérimentation a été généré avec les caractéristiques suivantes:

1. les tâches sont numérotées de 1 à 100 dans un ordre topologique;
2. le nombre de prédécesseurs de la tâche t est donné par tirage uniforme d'un nombre entier compris entre:
   - 1 et $t \div 2$ si $1 < t < 20$
   - 1 et 10 si $20 \leq t \leq 99$;
3. les prédécesseurs de t ($t \leq 99$) sont tirés sans remise parmi les tâches de numéros t -30 à t - 1; ceci afin d'obtenir un réseau "allongé";
4. le sommet 100 a pour prédécesseurs les tâches sans successeurs;
5. les durées des tâches 2 à 99 sont tirés uniformément entre 1 et 10;
6. pour une tâche donnée, chaque ressource a une probabilité 1/2 d'être prise.

Le tableau suivant donne pour chaque tâche: son numéro, sa durée, les ressources exigées, le nombre et la liste de ses prédécessseurs.

```
nombre de taches      : 100
nombre d'arcs         : 495
nombre de ressources : 4
```

1	0	0	0	0	0	0											
2	2	1	1	1	0	1	1										
3	2	0	0	0	1	1	1										
4	4	0	1	1	1	1	1										
5	2	1	1	0	0	1	2										
6	10	0	0	0	1	3	2	4	1								
7	1	0	0	1	0	2	2	6									
8	4	0	0	0	0	1	7										
9	3	1	1	0	1	3	7	5	2								
10	9	1	0	1	1	2	2	9									
11	7	0	0	0	1	1	3										
12	3	0	1	0	0	1	3										
13	4	0	0	0	1	6	10	8	7	3	11	6					
14	3	0	0	1	1	2	1	5									
15	7	0	0	0	1	2	10	3									
16	1	0	0	0	1	3	14	8	11								
17	2	1	1	1	0	7	1	6	2	3	8	14	7				
18	5	0	0	1	0	7	14	13	16	15	3	12	7				
19	4	0	0	0	1	5	6	16	8	4	12						
20	4	0	0	1	1	2	13	5									
21	6	1	0	1	0	10	2	18	17	4	10	7	8	20	13	16	
22	8	1	0	1	0	7	11	1	2	5	3	18	14				
23	9	1	0	0	0	10	13	22	'11	16	4	17	14	9	6	3	
24	5	0	1	1	0	8	8	13	10	19	6	3	5	23			
25	6	1	1	1	1	8	19	24	21	17	1	16	14	4			
26	2	1	0	0	0	4	22	24	1	17							

27	7	0	1	0	0	2	7	23										
28	8	1	0	0	1	7	3	19	6	11	22	16	23					
29	1	1	1	0	0	7	2	26	23	7	12	28	19					
30	8	1	1	1	0	2	11	10										
31	7	1	0	1	0	10	20	27	9	12	7	17	29	30	6	5		
32	7	1	0	1	0	7	5	17	3	10	16	12	25					
33	2	1	0	1	1	10	3	32	9	6	19	10	15	20	4	14		
34	3	1	1	1	0	4	18	14	17	30								
35	9	0	0	1	0	7	10	33	7	24	32	11	12					
36	2	1	1	1	0	1	16											
37	1	1	1	0	0	4	18	11	35	14								
38	6	0	1	0	1	3	32	19	12									
39	4	0	0	1	0	5	12	21	20	36	28							
40	1	0	0	0	1	2	27	36										
41	1	1	0	1	1	6	20	34	26	11	37	27						
42	5	0	1	1	1	3	28	13	15									
43	10	0	0	0	1	10	21	32	25	28	27	29	22	18	24	37		
44	7	1	1	1	0	3	31	14	40									
45	8	0	0	0	1	7	31	34	23	21	27	30	33					
46	7	1	0	0	0	4	29	25	33	32								
47	4	1	1	1	1	5	18	23	41	19	20							
48	10	0	0	0	0	2	40	26										
49	8	1	0	0	1	7	24	28	33	32	21	39	48					
50	5	1	0	0	1	3	35	46	41									
51	2	1	0	0	1	5	38	27	37	46	26							
52	5	0	1	1	1	10	48	40	26	49	30	25	29	45	32	44		
53	5	1	1	0	0	1	43											
54	7	0	0	0	0	4	29	35	45	33								
55	6	1	1	1	1	2	31	32										
56	3	1	0	1	1	2	47	37										
57	9	1	1	1	1	4	56	40	38	30								
58	7	1	0	1	1	9	35	50	42	49	34	55	38	57	30			
59	10	0	0	0	1	8	40	49	44	42	30	50	32	37				
60	6	1	1	0	0	1	47											
61	7	0	0	0	0	9	55	35	49	54	56	38	31	33	43			
62	10	0	1	0	1	6	61	60	53	37	46	39						
63	9	1	1	0	0	9	42	54	46	41	36	33	37	38	51			
64	2	1	1	1	0	9	57	62	51	61	47	50	52	37	46			
65	8	0	1	1	1	5	49	51	55	60	41							
66	9	0	0	1	0	2	65	43										
67	4	0	0	0	0	6	40	47	55	42	46	66						
68	4	0	0	0	0	3	45	53	65									
69	8	1	1	0	1	9	41	62	51	57	50	52	43	55	42			
70	6	0	0	1	0	7	67	43	57	45	42	40	51					
71	1	1	1	0	1	8	55	49	62	59	56	70	53	50				
72	4	1	1	0	0	10	48	57	43	67	42	45	59	44	63	62		
73	7	1	0	1	0	8	67	49	47	45	55	46	66	68				
74	9	1	1	0	0	2	58	70										
75	3	0	0	1	0	9	60	71	73	45	64	63	47	66	74			
76	4	0	1	0	1	1	52											
77	3	0	0	0	0	4	55	65	59	69								
78	4	0	0	1	0	8	60	72	75	74	62	48	50	53				
79	10	1	0	0	0	3	73	69	59									
80	7	1	0	1	0	6	68	60	72	71	50	61						
81	10	1	0	0	1	6	51	74	72	80	79	62						
82	9	1	1	0	1	1	75											

83	10	1	0	0	1	6	67	82	80	76	57	55							
84	5	0	1	1	0	9	79	74	77	70	81	57	64	76	63				
85	10	1	1	1	0	10	64	76	78	56	84	57	72	69	63	81			
86	4	1	1	1	1	6	62	57	61	63	77	85							
87	1	0	1	0	1	3	85	79	71										
88	9	1	1	1	1	2	58	67											
89	1	1	1	0	0	8	77	74	59	64	66	83	84	68					
90	1	0	0	0	1	4	82	74	60	67									
91	3	1	1	0	1	2	83	84											
92	9	0	1	0	0	3	86	91	75										
93	2	0	1	0	1	1	72												
94	5	0	0	0	0	9	72	80	82	84	88	67	81	77	86				
95	7	1	0	0	0	1	87												
96	2	0	1	1	1	8	70	91	92	85	81	76	67	84					
97	7	0	0	1	0	6	92	89	88	81	93	76							
98	8	1	0	1	1	8	96	80	90	84	97	70	71	83					
99	4	0	1	1	1	10	83	82	98	96	73	76	70	91	79	87			
100	0	0	0	0	0	3	94	95	99										

Le graphe de simultaneïté de ce projet est conséquent. En générant des stables de manière aléatoire sur ce stable, on a pu déterminer un stable de valeur 340. Cette valeur sera donc prise comme borne inférieure de la durée optimale de ce projet.

Bibliographie

E.H.L AARTS and P.J.M van LAARHOVEN Statistical Cooling: a general approach to combinatorial optimisation problem
Philips Journ Res 40 (1985) 193-226

K. R. Baker Introduction to sequencing and scheduling
John Wiley & Sons, Inc 1974

E. Bonomi, J.L. Lutton the N-city travelling salesman problem: statistical mechanics and Metropolis algorithm
SIAM review vol 26 n° 4 oct 1984

J. Carlier et P. Chretienne Les problèmes d'ordonnancement: un domaine ouvert RAIRO vol 16 n°3 aout 1982 p 176 - 217

M.R. Garey and D.S. Johnson Computers and intractability: a guide of the theory of NP-completeness W.H Freeman San Francisco 1979

M. Gondran et M. Minoux Graphes et Algorithmes
Collection de la direction des études et recherches d'EDF
Eyrolles 1980

E.L Lawler Recent results in the theory of machine scheduling mathematical programming: the state of the art
Bonn 1982 p 202-234 Springer Verlag

E. Maurel, D. Roux, D. Dupont Techniques opérationnelles d'ordonnancement, Collection de la direction des études et recherches d'EDF Eyrolles 1977

F. Premti méthodes stochastiques dans les problèmes de placement thèse de 3ième cycle - USMG 1983

Références bibliographiques

CA J. Carlier ordonnancements a contraintes disjonctives
RAIRO vol 12 n°4 nov 1978 p 335-353

DI R.P Dilworth A decomposition theorem for partially ordered set
Ann of Math 51 (1950) p 161-166

FU D.R Fulkerson A note on Dilworth's theorem for partially ordered sets
Proc Amer Math Soc 7 (1956) 701-702

GM M. Gondran et M. Minoux Graphes et Algorithmes
Collection de la direction des études et recherches d'EDF
Eyrolles 1980 p 165

GO Golumbic Algorithmic Perfect Graphs
Academic press New York 1980

KI S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi Optimisation by simulated annealing Res Rep RC 9335, IBM Thomas J Watson Center
Yorktown Heights NY 1982

LO L. Lovasz normal hypergraphs and the perfect graph conjecture
Discrete Math 2 (1972) p 253-267

ME N. Metropolis, A. et M. Rosenbluth, A. et E. Teller Equation of state calculations by fast computing machines J. Chem Phys 21 (1953) p 1087 - 1092

ST F. Sterboule et D. Wertheimer Comment construire un graphe Pert minimal RAIRO vol 14 n°1 fevr 1980 p 85-98