Ordonnancement: Modélisation et Algorithmes

1. Fonction de l'ordonnancement

L'ordonnancement est l'un des problèmes centraux, il appartient aux trois fonctions techniques de l'industrie: études, production et maintenance. Sa mission est de :

- Prévoir la chronologie du déroulement des tâches.
- Optimiser l'utilisation des moyens nécessaires, et les rendre disponibles.
- Lancer les travaux au moment choisi.
- Contrôler l'avancement et la fin des tâches, et prendre en compte les écarts entre prévisions et réalisations.

L'ordonnancement passe par trois étapes :

- La planification : qui vise à déterminer les différentes opérations à réaliser, les dates correspondantes, et les moyens matériels et humains à y affecter.
- L'exécution : qui consiste à la mise en œuvre des différentes opérations définies dans la phase de planification.
- Le contrôle : qui consiste à effectuer une comparaison entre planification et exécution, soit au niveau des coûts, soit au niveau des dates de réalisation.

1.1. Théorie de l'ordonnancement

La théorie de l'ordonnancement est une branche de la recherche opérationnelle qui s'intéresse au calcul de dates d'exécution optimales de tâches. Pour cela, il est très souvent nécessaire d'affecter en même temps les ressources nécessaires à l'exécution de ces tâches.

1.2. Problème d'ordonnancement

Un problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises. Cette programmation se fait alors en vue d'optimiser un ou plusieurs critères tels que la minimisation de la date d'achèvement de la réalisation, ou l'optimisation de l'utilisation des ressources sous

contraintes de durée. Les problèmes d'ordonnancement apparaissent, par exemples, en informatique (les tâches sont alors des processus informatiques et les ressources sont des processeurs et de la mémoire) ou dans l'industrie (dans un atelier de production, les tâches sont des traitements à appliquer à des pièces et les ressources sont constituée par les machines et le personnel).

1.3. Ordonnancement

Un ordonnancement constitue une solution au problème d'ordonnancement. Il est défini par le planning d'exécution des tâches (« ordre » et « calendrier ») et d'allocation des ressources et vise à satisfaire un ou plusieurs objectifs. Un ordonnancement est très souvent représenté par un diagramme de Gantt.

2. Concepts de base de l'ordonnancement

Les principaux éléments qui caractérisent un problème d'ordonnancement sont : les tâches, les ressources, les différentes contraintes s'y rapportent et les objectifs.

2.1. Tâche

Une tâche est une entité élémentaire localisée dans le temps par une date de début et/ou de fin, dont la réalisation nécessite une durée, et qui consomme un moyen (utilise une ou plusieurs ressources) selon une certaine intensité.

Deux types de tâches sont distingués:

- les tâches ***morcelables*** (*préemptives*) qui peuvent être exécutées en plusieurs fois facilitant ainsi la résolution de certains problèmes,
- les tâches ***non morcelables*** (*indivisibles*) qui sont exécutées en une seule fois et ne peuvent pas être interrompues avant qu'elles soient complètement terminées.

Chaque tâche *i* qui correspond à la réalisation d'un job *j* est alors caractérisée par :

- $\mathbf{t}_{ijk}$ **:** sa date de début d'exécution sur la machine *k.*
- $\mathbf{r}_{ijk}$: sa date de disponibilité ou date de début au plus tôt sur la machine *k.*
- $\mathbf{p}_{ijk}$: son temps d'exécution ou sa durée opératoire sur la machine *k.*
- $\mathbf{C}_{ijk}$: sa date de fin d'exécution sur la machine *k.*
- $\mathbf{d}_{ijk}$: sa date de fin au plus tard sur la machine *k.*

Ainsi, pour qu'un ordonnancement soit réalisable, la condition suivante est nécessaire :

$\forall\ i \in I$, $r_{ijk} \leq t_{ijk} \leq C_{ijk} \leq d_{ijk}$ Où I représente l'ensemble de tâches à ordonnancer.

2.2. Les ressources

La ressource est un moyen technique ou humain, disponible en quantité limitée, utilisé pour réaliser une tâche. Plusieurs types de ressources sont à distinguer:

- Les ressources ***renouvelables***, qui, après avoir été allouées à une tâche, redeviennent disponibles et qui peuvent être réutilisées (machines, personnel, etc.).
- Les ressources ***consommables***, qui, après avoir été allouées à une tâche, ne sont plus disponibles, et sont donc épuisées (argent, matières premières, etc.).
- Les ressources partageables qui peuvent être partagées entre plusieurs tâches.

Ces ressources peuvent être classées d'une autre manière :

- Les ressources de type ***disjonctif*** qui ne peuvent exécuter qu'une opération ou une tâche à la fois.
- Les ressources de type ***cumulati***f qui peuvent exécuter plusieurs opérations simultanément.

2.3. Les contraintes

Les contraintes représentent les conditions à respecter lors de la construction de l'ordonnancement pour qu'il soit réalisable. Elles rendent les problèmes d'ordonnancement plus difficiles car il faut les respecter lors de la résolution de ces problèmes.
On distingue :

- **des contraintes temporelles**

  - les contraintes de temps alloué, issues généralement d'impératifs de gestion et relatives aux dates limites des tâches (délais de livraisons, disponibilité des approvisionnements) ou à la durée totale d'un projet.
  - les contraintes de cohérence technologique, ou contraintes de gammes, qui décrivent des relations d'ordre relatif entre les différentes tâches.

- **des contraintes de ressources**

  - les contraintes d'utilisation de ressources qui expriment la nature et la quantité des moyens utilisés par les tâches, ainsi que les caractéristiques d'utilisation de ces moyens.

  - les contraintes de disponibilité des ressources qui précisent la nature et la quantité des moyens disponibles au cours du temps. Toutes ces contraintes peuvent être formalisées sur la base des distances entre débuts de tâches ou potentiels.

2.4. Les objectifs

Dans la résolution d'un problème d'ordonnancement on peut choisir entre deux grands types de stratégies, visant respectivement à l'optimalité des solutions, ou plus simplement à leur admissibilité.

L'approche par optimisation suppose que les solutions candidates à un problème puissent être ordonnées de manière rationnelle selon un ou plusieurs critères d'évaluation numériques, construits sur la base d'indicateurs de performances. On cherchera donc à minimiser ou maximiser de tels critères.

- **liés au temps** :

  - le temps total d'exécution ou le temps moyen d'achèvement d'un ensemble de tâches
  - le stock d'en-cours de traitement
  - différents retards (maximum, moyen, somme, nombre, etc.) ou avances par rapport aux dates limites fixées.

- **liés aux ressources** :

  - la quantité totale ou pondérée de ressources nécessaires pour réaliser un ensemble de tâches
  - la charge de chaque ressource ;
  - une énergie ou un débit ;
  - coûts de lancement, de production, de transport, etc., mais aussi aux revenus, aux retours d'investissements.

3. Différents problème d'ordonnancement

Les problèmes d'ordonnancement se divisent en deux grandes catégories selon le nombre d'opérations nécessaires à la réalisation de chaque travail:

- La première catégorie :
  - Machine unique
  - Machines dédiées
  - Machines parallèles

- la deuxième catégorie **:** problème d'atelier
  - Ateliers à cheminement unique (Flow Shop)
  - Ateliers à cheminements multiples (Job Shop)
  - les ateliers à cheminement libre (open shop)

3.1. La première catégorie

La première catégorie se subdivise à tour en plusieurs types de problèmes, en fonction de la configuration de machines considérée :

- **Machine Unique :** Dans ce cas, l'ensemble des tâches à réaliser est fait par une seule machine. Les tâches alors sont composées d'une seule opération qui nécessite la même machine.
- **Machine dédiée :** Plusieurs machines, chacun étant spécialisée pour l'exécution de certains travaux**.**
- **Machines parallèles :** Dans ce cas, on dispose d'un ensemble de machines identiques pour réaliser les travaux. Les travaux se composent d'une seule opération et un travail exige une seule machine. L'ordonnancement s'effectue en deux phases : la première phase consiste à affecter les travaux aux machines et la deuxième phase consiste à établir la séquence de réalisation sur chaque machine.

3.2 .La deuxième catégorie (problème d'atelier)

Regroupe les problèmes pour lesquels chaque travail nécessite plusieurs opérations. Ils sont généralement spécifiés par la donnée de *m* machines et de *n* travaux composés chacun de *m* opérations ; chaque opération devant être exécutée par une machine différente. Trois

sous-classes de problèmes sont alors différenciées selon le mode de passage des opérations sur les différentes machines, à savoir :

- **Flow Shop**

  Les ateliers de type « flow-shop » pour lequel la ligne de fabrication est constituée de plusieurs machines en série ; toutes les opérations de toutes les tâches passent par toutes les machines dans le même et unique ordre .Ce type d'atelier est dit à cheminement unique, cas d'une chaîne de fabrication). L'un des objectifs principaux de flow-shop est de trouver une séquence des tâches en main qui respecte un ensemble de contraintes et qui minimise le temps total de production.

- **Job Shop**

  Dans les ateliers de type « job-shop », les opérations sont réalisées selon un ordre total bien déterminé, variant selon la tâche à exécuter. Ce type d'atelier est nommé aussi atelier à cheminements multiples, (cas de l'atelier de production traitant plusieurs produits).

- **Open shop**

  Chaque produit à traiter doit subir un ensemble d'opérations sur un ensemble de machines, mais dans un ordre totalement libre.

4. Modélisation des problèmes d'ordonnancement

4.1. Classification

Pour présenter un problème d'ordonnancement, nous adoptons la notation proposée par Graham et al. permettant de distinguer les classes des problèmes d'ordonnancement. Ce formalisme contient trois champs séparés par des « slashs » (α / β/ δ).

- **Le premier champ α**

  ll représente *l'environnement des ressources* et est spécifié par concaténation de deux éléments : α =α*1*α*2*

  - Le paramètre α*1* représente la configuration de machines utilisées:

    (α*1* ∈ {∅, *P, Q, R, F, O, J*})

    - α*1* = ∅ : une seule machine est utilisée.
    - α*1* = *P* : plusieurs machines identiques sont disponibles. (i.e. les ressources sont composées de machines travaillant suivant la même cadence, disposées en parallèle et pouvant exécuter tous les travaux).
    - α*1* = *Q* : plusieurs machines parallèles uniformes sont disponibles. (i.e. les cadences des machines sont différentes (selon un facteur de proportionnalité), mais restent indépendantes des travaux.
    - α*1* = *R* : plusieurs machines indépendantes non liées sont disponibles. (i.e. les cadences des machines sont différentes et dépendent des travaux exécutés).
    - α*1* = *F* : plusieurs machines dédiées fonctionnant en flow-shop. (i.e. les travaux sont décomposés en plusieurs opérations qui doivent être exécutées sur l'ensemble des machines, celles-ci étant disposées en série pour un même routage).

- $\alpha 1 = O$: plusieurs machines dédiées fonctionnant en open-shop. (i.e. les travaux sont décomposés en plusieurs opérations qui doivent être exécutées sur l'ensemble des machines sans restriction sur le routage des travaux).
- $\alpha 1 = J$: plusieurs machines dédiées fonctionnant en job-shop. (i.e. les travaux sont décomposés en plusieurs opérations qui doivent être exécutées sur l'ensemble des machines, mais peuvent avoir des routages différents).

▪ Le paramètre $\alpha 2$ permet de préciser le nombre de machines composant l'atelier; il peut être égal à vide ou à un entier m. Dans le premier cas, cela signifie que le nombre de machines est quelconque. Dans le deuxième cas, cela signifie que l'atelier est composé de m machines ($m > 0$).

➢ **Le deuxième champ β**

Il représente *les contraintes et les caractéristiques du système*. Il est formé de huit sous champs, $\beta = \beta 1 \beta 2 \beta 3 \beta 4 \beta 5 \beta 6 \beta 7$.

▪ $\beta 1 \in \{\varnothing, pmtn\}$ permet de préciser le mode d'exécution. $\beta 1 = \varnothing$ indique le mode sans préemption et $\beta 1 = pmtn$ indique le mode avec préemption.

▪ $\beta 2 \in \{\varnothing, prec, tree, chain\}$ précise un type de précédence entre travaux, c'est-à-dire le fait qu'un travail doit être exécuté avant un autre. La valeur $\varnothing$ indique que les travaux sont indépendants. Les valeurs *prec, tree, chain* indiquent l'existence, respectivement, d'une relation de précédence générale, d'une relation de précédence sous forme d'arbre et d'une relation de précédence sous forme de chaîne.

▪ $\beta 3 \in \{\varnothing, r_j\}$ décrit les dates de disponibilité (i.e. dates au plus tôt) des différents travaux dans le système. Ces dates peuvent être identiques et égales à zéro pour tous les travaux ($\beta 4 = \varnothing$) ou différentes suivant les travaux ($\beta 4 = r_j$).

- $\beta 4 \in \{\varnothing, p_j = p, \underline{p} \leq p \leq \overline{p}\}$ détaille les durées opératoires des différents travaux.

  Ces durées peuvent être fonction de la machine. Différentes restrictions peuvent également être considérées pour simplifier certains problèmes.

  - $\beta 4 = \varnothing$: les travaux ont des durées opératoires arbitraires.
  - $\beta 4 = p_j = p$: tous les travaux ont une durée opératoire égale à p.
  - $\beta 4 = \underline{p} \leq p \leq \overline{p}$: les durées opératoires des travaux sont comprises entre $\underline{p}$ et $\overline{p}$.

- $\beta 5 \in \{\varnothing, d_j, \tilde{d}_j\}$ indique les éventuelles dates d'échéance (ou dates au plus tard) des travaux.

  - $\beta 5 = \varnothing$: les travaux n'ont pas de date d'échéance.
  - $\beta 5 = d_j$: chaque travail a une date d'échéance de fin d'exécution sous peine de pénalisation.
  - $\beta 5 = \tilde{d}_j$: chaque travail a une date d'échéance impérative (date limite) qu'il faut absolument respecter.

- $\beta 6 \in \{\varnothing, s, s_i, s_{ij}, nwt\}$ permet de spécifier des contraintes temporelles sur les enchaînements de travaux. Ces contraintes sont très souvent introduites afin de mieux représenter les problèmes réels. Il est parfois nécessaire de considérer un temps improductif entre l'exécution de deux travaux différents sur une même machine pour représenter les changements et les réglages d'outils. Ces temps de changement peuvent être constants (s), fonction du nouveau travail (s_i) ou bien fonction de l'enchaînement des deux travaux (s_{ij}).

  Pour le cas $\beta 6 = nwt$: toutes les opérations d'un travail soient exécutées sans temps d'attente (no-wait).

- Enfin, le paramètre $\beta 7 \in \{\varnothing, M_j\}$ indique, dans le cas de machines parallèles, des restrictions sur la polyvalence des machines. L'ensemble M_j représente l'ensemble des machines capables de réaliser le travail J_j. Lorsque $\beta 7$ est vide, toutes les machines sont capables d'exécuter tous les travaux.

➢ **Le troisième champ** γ

Il spécifie le critère à optimiser. Les critères les plus utilisés sont

- $\gamma = C_{max}$: makespan. Date de sortie du système (le dernier travail).
  ($C_{max} = \max_j C_j$ où C_j est la date de fin d'exécution du travail J_j).
- $\gamma = \Sigma C_j$ (ou C^*) : somme des dates de fin d'exécution. La date C_j peut être pondérée et un deuxième critère pourra être défini $\gamma = \Sigma w_j C_j$, où w_j est le poids associé au travail J_j.
- $\gamma = L_{max}$: décalage temporel maximal. Ce critère mesure la plus grande violation des dates d'échéance ($L_{max} = \max_j \{L_j = C_j - d_j\}$ où d_j est la date de fin au plus tard).
- $\gamma = \Sigma U_j$: somme du nombre de travaux terminés avec retard ($U_j = 1$ si le travail J_j est terminé après son échéance au plus tard). Un U_j peut être pondéré et un deuxième critère pourra être $\gamma = \Sigma w_j U_j$, où w_j est le poids associé au travail J_j.

Un sous-ensemble d'ordonnancements est dit *dominant* par rapport à un critère si ce sous-ensemble contient au moins un ordonnancement optimal relativement à ce critère.

4.2. Modélisation

La modélisation, est en général, une étape très importante dans la résolution d'un problème d'ordonnancement. C'est une écriture simplifiée de toutes les données du problème permettant d'en traduire tous les détails pour mieux représenter la réalité des choses.

Il existe trois méthodes pour modéliser l'ordonnancement : le diagramme de Gantt, la méthode MPM (Méthode des potentiels Métra), le PERT (Program Research Technic).

4.2.1. Le Diagramme de Gantt

C'est une méthode très ancienne puisque datant de 1918 et pourtant encore très répandue mais sous des formes et sur des applications résolument modernes. Consiste à déterminer la meilleure manière de positionner les différentes tâches d'un projet à exécuter.

✓ **Principe**

Ce type de diagramme a été mis au point par un américain Henry Gantt. On représente au sein d'un tableau, en ligne les différentes tâches et en colonne les unités de temps (exprimées en mois, semaines, jours, heures…)

La durée d'exécution d'une tâche est matérialisée par un trait au sein du diagramme.

- ✓ **Réalisation.**

Les différentes étapes de réalisation d'un diagramme de Gantt sont les suivantes :

Première étape : On détermine les différentes tâches (ou opérations) à réaliser et leur durée.

Deuxième étape : on définit les relations d'antériorité entre tâches.

Troisième étape : on représente d'abord les tâches n'ayant aucune antériorité, puis les tâches dont les tâches antérieures ont déjà été représentées, et ainsi de suite…

Quatrième étape : on représente par un trait parallèle en pointillé à la tâche planifiée la progression réelle du travail.

Exemple

Temps / Tâche	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
A													
B													
C													
D													
E													

Remarques

- Chaque colonne représente une unité de temps.
- Les durées d'exécution prévues des tâches sont représentées par un trait épais. (4 unités de temps pour C).
- Les contraintes de succession se lisent immédiatement.
  - Les tâches B et C succèdent à la tâche A.
  - D succède à B.
  - E succède à D.

- On peut alors déterminer **le chemin critique** : qui est formé d'une succession de tâches, sur le chemin le plus long en terme de durées. Il est appelé chemin critique car tout retard pris sur l'une des tâches de ce chemin, entraîne du retard dans l'achèvement du projet. (Chemin critique : A, B, D, E).

Avantages

- Permet de déterminer la date de réalisation d'un projet.
- Permet d'identifier les marges existantes sur certaines tâches (avec une date de début au plus tôt et une date au plus tard).
- La date au plus tard de début d'une tâche, la date à ne pas dépasser sans retarder l'ensemble du projet.

Inconvénient

Ne résoudre pas tous les problèmes, en particulier si l'on doit planifier des fabrications qui viennent en concurrence pour l'utilisation de certaines ressources.

4.2.2. La méthode des potentiels métra (MPM)

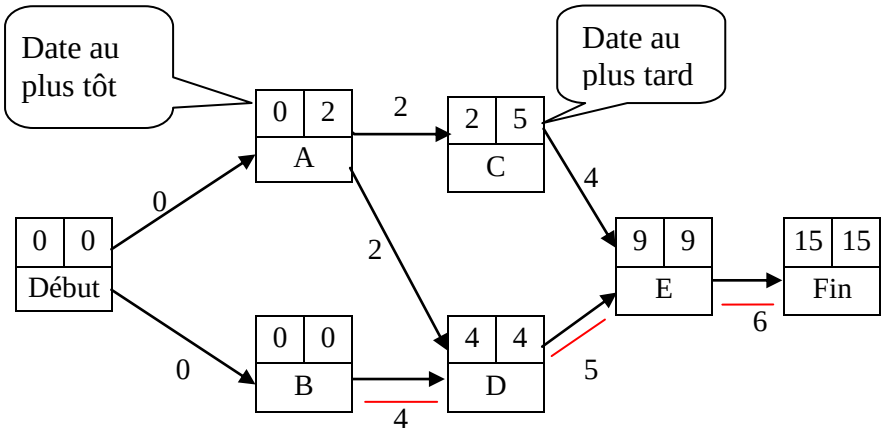
La méthode des potentiels métra (MPM) a été développée par une équipe de chercheurs français.

Principe.

- Les tâches sont représentées par des sommets et les contraintes de succession par des arcs.
- Chaque tâche est renseignée par sa date début au plus tôt et sa date début au plus tard
- A chaque arc est associée une valeur numérique, qui représente la durée de l'opération.

Exemple

Tâche	Durée	Tâches antérieures
A	2	———
B	4	———
C	4	A
D	5	A, B
E	6	C,D



Remarques

- La date de début au plus tôt d'une tâche est obtenue en cumulant la durée des tâches qui précèdent sur la séquence la plus longue.
- On initialise le sommet DEBUT avec une date au plus tôt = 0.

  Pour tous les prédécesseurs *i* de *j* :

  Date au plus tôt de la tache j= Max (date au plus tôt de i + Durée i)

- Les dates au plus tard : dates à laquelle doivent être exécutées les tâches sans remettre en cause la durée optimale de fin du projet.
- On initialise à l'étape terminale, le dernier sommet par la date au plus tard =date au plus tôt.

  Pour tous les successeurs *j* de *i* :

  Date au plus tard i = Min (Date au plus tard de j – durée i)

On peut alors déterminer **le chemin critique** : succession de tâches sur le chemin le plus long au sens des durées. Pour toutes les tâches du chemin critique, les dates au plus tôt et au plus tard coïncident. Chemin critique : B, D, E.

- La marge totale sur une tâche est le retard que l'on peut prendre dans la réalisation de cette tâche sans retarder l'ensemble du projet. Elle est obtenue, en faisant pour chaque tâche, la différence entre la date au plus tard de début d'une tâche et la date au plus tôt.
- La marge libre sur une tâche est le retard que l'on peut prendre dans la réalisation d'une tâche sans retarder la date de début au plus tôt de toute autre tâche qui suit.

Pour tous les successeurs j de i ; *Marge Libre de* $i = Min\ (T_j - T_i - D_{i\,j})$ où T_j est la date au plus tôt de la tâche qui suit la tâche considérée, T_i est la date de début au plus tôt de la tâche i et $D_{i\,j}$ est la durée qui sépare la tâche i de la tache j.

4.2.3. La méthode P.E.R.T (Program Evaluation and Research Task)

Le graphe PERT permet de visualiser la dépendance des tâches et de procéder à leur ordonnancement. On utilise un graphe de dépendances. Pour chaque tâche, on indique une date de début et de fin au plus tôt et au plus tard. Le diagramme permet de déterminer le chemin critique qui conditionne la durée minimale du projet.

Principe

Dans un graphe PERT :

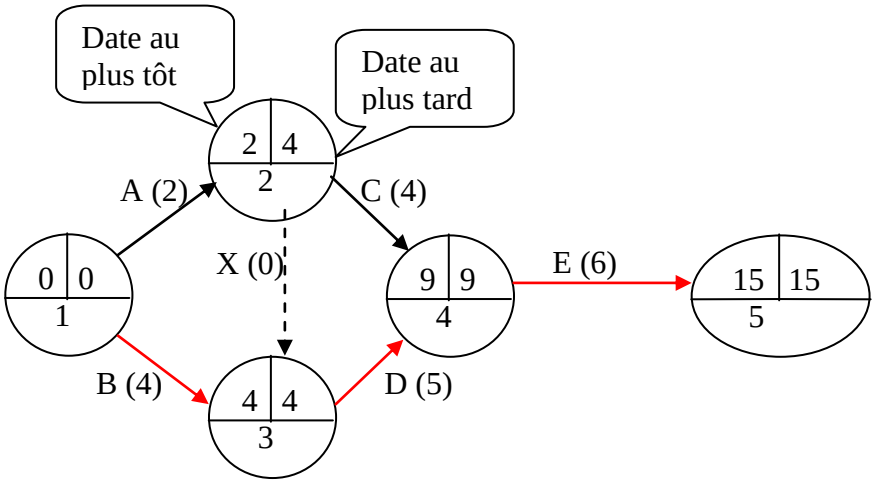
- Chaque tâche est représentée par un arc, auquel on associe un chiffre entre parenthèses qui représente la durée de la tâche.
- Entre les arcs figurent des cercles appelés « sommets » ou « événement » qui marquent l'aboutissement d'une ou plusieurs tâches. Ces cercles sont numérotés afin de suivre l'ordre de succession des divers évènements.

Réalisation

Pour construire un graphe PERT, on utilise la méthode de décomposition en niveaux :

- On détermine les tâches sans antécédents, qui constituent le niveau 1.
- On identifie ensuite les tâches dont les antécédents sont exclusivement du niveau 1. Ces tâches constituent le niveau 2, et ainsi de suite…

Exemple



Remarques

- Il a été nécessaire d'introduire une tâche fictive de durée égale à 0, pour représenter la relation d'antériorité entre A et D
- Le cumul des tâches composant la séquence la plus longue (B, D, E) permet de déterminer la date au plus tôt de réalisation du projet. Cette succession de tâches constituent le chemin critique.

- **Date au plus tôt :** on initialise la date au plus tôt du premier sommet à 0 :

  T_1 = 0 : Désigne la date au plus tôt du sommet 1.

  Pour tous les prédécesseurs i de j :

  $T_j = Max\ (T_i + Durée_i)$

- **Date au plus tard :** on initialise la date au plus tard du dernier sommet avec sa date au plus tôt.

  $T_n^* = T_n$ (T_n^* désigne la date au plus tard du sommet *n*, T_n désigne la date au plus tôt du sommet *n*).

  Pour tous les successeurs j de i :

  $$T_i^* = Min\left(T_j^* - Durée_i\right)$$

- **Marge totale :**

  $T_{i,j}$: durée de la tâche entre les sommets i et j.

  $$Marge\ totale_{i,j} = T_j^* - T_i - T_{i,j}$$

- **Marge Libre :**

  $$Marge\ libre_{i,j} = T_j - T_i - T_{i,j}$$

- Sur le chemin critique, les marges totales des différentes tâches sont nulles.
- Pour toute tache, sa marge libre est toujours inférieure ou égale à sa marge totale.

5. Etude de quelques problèmes

Le but de cette partie est de présenter quelques résultats de base sur les problèmes à une et *m* machines. Ces problèmes peuvent sembler très spécifiques. Pourtant leur résolution est à la base de problèmes plus généraux, en particulier, quand il apparaît qu'une ressource est dominante.

5.1 Ordonnancement sur une machine

- **Problème 1 \ \ C_{max}**

  $C_{max} = \max C_i$

  Objectif : Minimiser la durée totale de l'ordonnancement.

  Toute séquence nous donne une solution optimale.

- **Problème 1 \ \ $\sum C_i$**

  Objectif : Minimiser la durée totale de l'ordonnancement.

  - ✓ Attente moyenne des clients dans une file.
  - ✓ Stocks d'encours devant la ressource.

  Supposons qu'on a 4 taches à ordonnancer A, B, C, D de durées respectivement P_A, P_B, P_C et P_A. Prenons la séquence ABCD

  Alors on aura :

  $C_A = P_A$

  $C_B = P_A + P_B$

  $C_C = P_A + P_B + P_C$

  $C_D = P_A + P_B + P_C + P_D$

  $C_A + C_B + C_C + C_D = 4P_A + 3P_B + 2P_C + P_D$

  Donc La règle SPT (Shortest Processing Time) séquençant les taches par durée opératoire croissante est une séquence qui vérifie la dominance, par conséquent elle est optimale pour 1 \ \ $\sum Ci$

- **Problème 1 \ r_i \ C_{max}**

  Toutes les taches ne sont pas disponibles dès le début de l'ordonnancement mais au fur et à mesure.

  - ✓ Date ri (release date) d'arrivée / disponibilité.
  - ✓ Des temps d'inactivité (idle time) peuvent paraitre sur la ressource.

Exemple

	A	B	C	D
p_i	5	2	1	3
r_i	0	9	1	8

La séquence ABCD donne C_{max} =15

La séquence ACDB donne C_{max} =13

La meilleure séquence c'est celle qui minimise le temps total d'inactivité. Ce problème est résolu en temps polynomial en ordonnant les opérations par ordre croissant sur les dates de disponibilité

- **Problème 1 \ r_i \ $\sum C_i$**
  - Ce problème est NP-difficile,

Algorithme de Liste

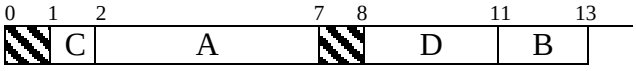
- ✓ Priorité sur les taches : liste L
- ✓ séquençage glouton des taches : Si la ressources est libre, séquencer la première tache disponible de la liste.
- ✓ Principe glouton : ne pas laisser la ressource inoccupée si les taches sont disponibles
- ✓ La liste sert à arbitrer lorsque plusieurs taches sont disponibles en même temps

Exemple

	A	B	C	D
p_i	5	2	1	3
r_i	0	9	1	8

Prenons une liste de priorité SPT, L = CBDA. En appliquant l'algorithme de liste on aura $\sum C_i$ = 35.

Remarquons que la séquence CADB donne $\sum C_i$ = 33



Remarques

- Pour l'exemple donné, quelque soit la liste de priorité, le résultat de l'algorithme reste le même. Donc il n'arrive jamais à une solution optimale.
- Dans la séquence optimale il peut être nécessaire de laisser la ressource inoccupée même si des taches sont disponibles
- Tout algorithme de liste peut être arbitrairement mauvais

Modèle mathématique : Une formulation PLNE

Variable de décision :

C_i : date de fin de la $i^{ème}$ tache

Objectif : $\min \sum_i C_i$

Contraintes :

$$C_i \geq r_i + p_i$$

$$C_i \geq C_j + p_i \quad \text{XOR} \quad C_j \geq C_i + p_j \quad \forall i,j$$

Cette contrainte disjonctive peut être remplacée par :

Variable de décision :

$$x_{ij} = \begin{cases} 1 \ si \ i \ avant \ j \\ 0 \ sinon \end{cases}$$

Big M

$$C_i \geq C_j + p_i - M x_{ij} \quad \forall i,j$$

$$x_{ij} + x_{ji} = 1 \quad \forall i,j \quad i \neq j$$

Heuristique

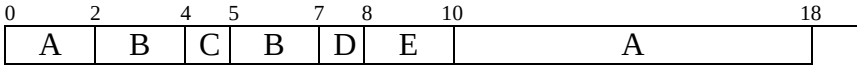
Relaxation préemptive : Relaxation du problème en autorisant la préemption autrement dit, une tache peut être exécutée en plusieurs fois (en morceaux).

La règle SRPT (Shortest Remaining Processing Time) ordonnançant à chaque instant la tache disponible de plus petit temps restant est optimale pour $1 \setminus r_i$, $pmtn \setminus \sum C_i$.

Exemple

	A	B	C	D	E
p_i	10	4	1	1	2
r_i	0	2	4	6	8

En ordonnançant les taches selon la règle SRPT on aura le résultat suivant :

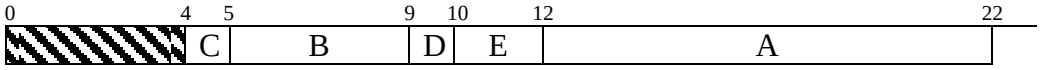


Donc $\sum C_i = 48$. Cette valeur représente une borne inférieure au problème $1 \setminus r_i \setminus \sum C_i$.

Algorithme de Stein

C'est un algorithme approché pour résoudre le problème $1 \setminus r_i \setminus \sum C_i$. Son principe consiste à ordonnancer les taches dans l'ordre de leurs dates de fin C_i dans l'ordonnancement du problème $1 \setminus r_i$, pmtn $\setminus \sum C_i$ selon la règle SRPT.

Pour l'exemple, la séquence obtenue est : CBDEA



Donc $\sum C_i = 58$

- **Problème $1 \setminus d_i \setminus \sum U_i$**

Algorithme de Hogdson

L'objectif est de minimiser le nombre de taches terminant en retard. On commence par le tri des taches selon les d_i croissant, suivi par la construction d'un ensemble de taches terminant sans retard. A chaque itération, on ajoute une tache, si cette tache fait du retard, on supprime de cet ensemble, la tache la plus longue.

Début

Ordonnancer les taches selon les d_i croissant

A ← ∅ /* A est l'ensemble des taches terminant sans retards */

Pour i = 1 à n faire

Si $P(A) + pi \leq di$ **Alors** /* $p(A) = \sum_{i \in A} p_i$ */

$A \leftarrow A + \{ i \}$

Sinon

Soit $j = argmax_{j \in A+\{i\}}\{p_j\}$

A ← A + { i } - { j }

Finsi

Finpour

Fin

- **Problème 1 \ prec \ C_{max}**

  - Tout algorithme de liste est optimal pour ce problème

- **Problème 1 \ prec, d_i \ T_{max}**

  Algorithme de Lawler

  L'objectif est de minimiser le plus grand retard effectué en présence des contraintes de précédence. Cet algorithme procède comme suit :

  Debut

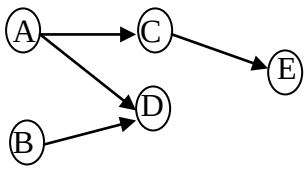
  /* t : représente la date fin de la tache qu'on va placer à chaque itération */

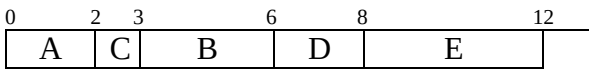
  1- Partons de la fin de l'ordonnancement ($t = \sum p_i$).
  2- Placer la tache i sans successeurs (non ordonnancé) qui minimise T_i.
  3- $t = t - p_i$
  4- S'il existe une tache non ordonnancé alors Aller à l'étape 2.

  Fin

  Exemple

	A	B	C	D	E
p_i	2	3	1	2	4
d_i	8	7	3	6	11





  $\sum p_i = 12$, le résultat de l'algorithme est la séquence ACBDE qui est optimale avec $T_{max} = 2$ pour D

5.2 Ordonnancement sur des machines parallèles

- **Problème Pm \ \ Cmax**

  - Ce problème est NP-difficile même si on a seulement deux machines.

Modèle mathématique : Une formulation PLNE

Problème purement d'affectation des taches aux ressources

Variables d'affectation :

$$a_{ik} = \begin{cases} 1 \ \ Si \ \ T_i \ effectuée \ sur \ la \ machine \ M_k \\ 0 \ \ Sinon \end{cases}$$

Contrainte d'affectation :

$\forall i \ \ \sum_k a_{ik} = 1$ /* Une tache est affectée à exactement une machine */

Date de fin sur la machine k :

$\sum_i p_i a_{ik}$ /* Somme des temps d'exécution des taches affectées à M_k */

Objectif : min $max_{k \sum_i p_i a_{ik}}$

- **Problème Pm \ pmtn \ Cmax**

Algorithme de Mac-Naughton

C'est un algorithme qui commence d'abord par le calcul d'une borne inférieure :

$LB = Max \left\{ Max_i\{p_i\}, \ \frac{1}{m}\sum_{i=1}^{n} p_i \right\}$ car la durée de l'ordonnancement est supérieure à la durée d'une tâche et à la somme des durées des tâches divisée par le nombre de machines (durée moyenne).

Une fois la borne *LB* est calculée, il passe à la phase d'affectation des taches une par une en commençant par la première machine. Si l'affectation de la tache courante provoque le dépassement de LB, cette tache sera interrompue et le reste de celle-ci sera affecté au début de la machine suivante et ainsi de suite jusqu'à l'affectation de la totalité des tâches. L'algorithme de Mac-Naughton construit donc un ordonnancement de durée égale à *LB*.

Algorithme Mac-Naughton

Début

$LB = Max\left\{Max_i\{p_i\},\ \frac{1}{m}\sum_{i=1}^{n} p_i\right\}$

Poser $t = 0$; k = 1 ;

Pour i = 1 à n

Si $t + p_i \leq LB$ **alors**

Affecter la tache i sur la machine k entre les instants t et $t + p_i$

$t \leftarrow t + p_i$

Sinon

Affecter la tache i sur la machine k entre les instants t et LB et sur la machine $k+1$ entre les instants 0 et $p_i - LB + t$

$k \leftarrow k + 1$

$t \leftarrow p_i - LB + t$

Fin Si

Fin Pour.

Fin.

- **Problème Pm \prec \ Cmax**

Pour chercher une solution approchée à ce problème on peut utiliser un algorithme de liste utilisant une liste de priorité tout en respectant les contraintes de précédence.

Algorithmes de liste

Soit $L = (J_1, J_2, \ldots, J_n)$ une liste des n jobs.

t := 0 ; NSJ := J ; /* L'ensemble de tous les jobs */

Tantque $NSJ \neq \varnothing$ **Faire**

Soit $PR(t)$ l'ensemble des jobs de NSJ prêts à l'instant t ;

Soit $ML(t)$ l'ensemble des machines libres à partir de t ;

k := Min{Card($PR(t)$, Card($ML(t)$)} ;

Exécuter les k premiers jobs de $PR(t)$ dans L sur k machines de $ML(t)$;

Supprimer les k jobs choisis de NSJ ;

t := Min {$u > t$ / une machine est libre à partir de u}

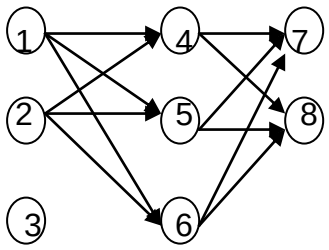
Fintantque

Exemple

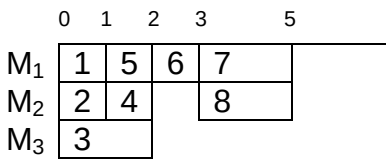
m=3 ;

p1 = p2 = p4 = p5 = p6 = 1 ; p3 = p7 = p8 = 2.

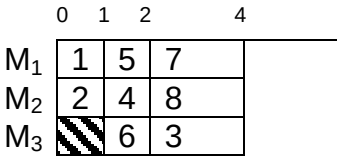
Soit L une liste quelconque.



Ordonnancement S(L)



Ordonnancement optimal



Deux propriétés fondamentales des ordonnancements de liste.

Propriété 1

Soit S(L) un ordonnancement de liste.

Si une machine est oisive sur un intervalle de temps $[t, t+\mathcal{E}]$ ($\mathcal{E} > 0$), alors pour tout job J_i ordonnancé après t, l'un des ascendants de J_i est en cours d'exécution sur $[t, t+\mathcal{E}]$.

Propriété 2

Soit *S(L)* un ordonnancement de liste d'un énoncé E de Pm / prec / C_{max}.

Soit C^* le délai minimum d'un ordonnancement de E.

Soit $C_{max}(S(L))$ le délai de S(L).

On a : $C_{max}(S(L)) \leq (2\text{-}1/m)\, C^*$.

5.3 Problèmes d'ateliers

- **Problème F2 \ \ C_{max}**

Les produits passent successivement sur une machine outil *A* et poste de finition *B*.

Objectifs : Finir au plus tôt les produits.

Algorithme de Johnson

Partitionner les jobs en deux sous ensembles :

- ✓ $A = \{i / a_i \leq b_i\}$ /* Jobs plus couts sur A */
- ✓ $B = \{i / a_i > b_i\}$ /* Jobs plus couts sur B */

Ordonnancer :

- Les jobs de A par a_i croissant
- Les jobs de B par b_i décroissant

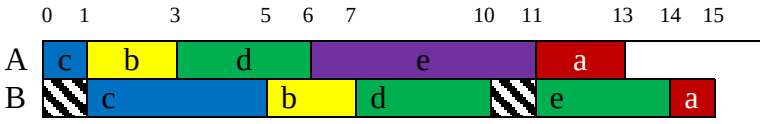
Exemple

	a	b	c	d	e
A	2	2	1	3	5
B	1	2	4	3	3

$A = \{b, c, d\}$ → $A = c, b, d$

$B = \{a, e\}$ → $B = e, a$

Donc $\sigma = c, b, d, e, a$



Le makespan = 15

- **Problème F3 \ \ Cmax**

Les produits passent successivement sur les machines *A, B* et *C*.

Objectifs : Finir au plus tôt les produits.

Ce problème est NP-difficile sauf si la machine B (étage intermédiaire) est dominée, c'est-à-dire $\max \{b_i\} \leq \min \{a_i\}$ ou $\max \{b_i\} \leq \min \{c_i\}$.

Si cette condition est vérifiée la solution optimale est obtenue par l'algorithme de Johnson légèrement modifié :

- ✓ Créer deux machines virtuelles MV_1 et MV_2 :
  - Les temps opératoires sur MV_1 sont $\{a_i + b_i\}$
  - Les temps opératoires sur MV_2 sont $\{b_i + c_i\}$
- ✓ Appliquer Johnson sur MV_1 et MV_2
- ✓ L'ordre trouvé est optimal sur les trois machines

- **Problème J2 \ \ Cmax**

Chaque job possède son propre ordre de passage et a au plus deux opérations.

Objectifs : Finir au plus tôt les produits.

Algorithme de Jackson

Partitionner les jobs en quatre sous ensembles :

- ✓ AB : Jobs passant d'abord sur la machine A.
- ✓ BA : Jobs passant d'abord sur la machine B.
- ✓ A : Jobs passant uniquement sur la machine A.
- ✓ B : Jobs passant uniquement sur la machine B.

Ordonnance les jobs selon les séquences :

- ✓ Machine A : {AB}(Johnson) {A} et {BA}
- ✓ Machine B : {BA}(Johnson) {B} et {AB}

Exemple

Les jobs {a, b} sont exécutés dans l'ordre *A*, *B*.
Les jobs {d, e} sont exécutés dans l'ordre *B*, *A*.

	a	b	c	d	e
A	3	4	1	3	1
B	1	5		1	2

$AB = \{a, b\} \xrightarrow{Johnson} AB = \{b, a\}$

$BA = \{d, e\} \xrightarrow{Johnson} AB = \{d, e\}$

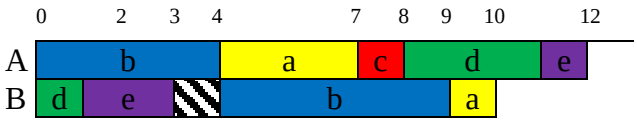
$A = \{a\}$.

$B = \emptyset$.

Donc on va Ordonnancer les jobs selon les séquences :

σ_A = b, a, c, d, e

σ_B = d, e, b, a



Le makespan = 12